Appendix B

Provisional Patent Application

6391-715:

Title: SELF MANAGEMENT PROTOCOL FOR A FLY-BY-WIRE SERVICE PROCESSOR

Invs: Karl Johnson
Walter A. Wallach
Dennis H. Smith
Carl G. Amdahl

The present invention provides a multi-processor diagnostic sub-system which implements a "fly-by-wire" environmental control system for a multi-processor, fault tolerant computer system, based on industry standard components but independent of any single implementation technology. Control of all physical resources (fans, power, etc.) is done by microprocessor control. Also, a low level non-volatile means is provided to log events and other system information used to diagnose failures.

The following documents are incorporated by reference and attached hereto:

1. *NetFRAME Wire Service Implementation User's Guide*, Version 1.0, February 29, 1997, pp. i-vii and pp. 1-47.

2. *NetFRAME Wire Service Implementation Raptor 8 Addendum*, Version 1.0, February 28, 1997, pp. i-iv and pp.1-7.

3. *NetFRAME Wire Service Implementation Command Reference*, Version 1.0, February 28, 1997, pp. i-vi and pp. 1-101.

4. *Raptor Wire Service Architecture*, Version 1.0, January 23, 1996, pp. 1-30.

Multiple Node Service Processor Network

A means is provided by which individual components of a system are monitored and controlled through a set of independent, programmable microcontrollers interconnected through a network. Further means are provided to allow access to the microcontrollers and the interconnecting network by software running on the host processor.

Fly-by-wire

A means is provided by which all indicators, push buttons and other physical control means are actuated via the multiple node service processor network. No indicators, push buttons or other physical control means are physically connected to the device which they control, but are connected to a microcontroller, which then actuates the control or provides the information being monitored.

Self-Managing Intelligence

A means is provided by which devices are managed by the microcontrollers in a multiple node service processor network by software running on one or more microcontrollers, communicating via the interconnecting network. Management of these devices is done entirely by the service processor network, without action or intervention by system software or an external agent.

Flight Recorder

A means is provided for recording system events in a non-volatile memory, which may be examined by external agents. Such memory may be examined by agents external to the network interconnecting the microcontrollers.

Replicated components: no single point of failure

A means is provided by which no single component failure renders the monitoring and control capability of the system inoperable.

Extension by serial or modem gateway

A means is provided allowing an external agent to communicate with the microcontrollers by extending the interconnecting network beyond the physical system.

The hardware environment is built around a self-contained network of microcontrollers. This distributed service processor continuously monitors and manages the physical environment of the machine (temperature, voltages, fan status). Intrapulse makes the physical environment self managing. For example, if a fan fails, Intrapulse detects this by continuously monitoring the speed of all fans, and increases the speed of the remaining fans in close proximity to the failed fan to maintain proper cooling, and signals an event to the system management software. Since the cooling system has twice the necessary cooling capacity, the system can continue to operate until a system administrator can schedule time to replace the failed fan. And, since the fans are all hot pluggable, the failed fan can be replaced without shutting down the operating system or denying service to clients. Following replacement, Intrapulse automatically detects that the fan is now working, and returns the other fans to their normal speed.

Since fans and power supplies are N+1 redundant, they all operate well below their rated speed or capacity, which increases their operating life and reduces the probability of their failing.

The basic philosophy behind the Intrapulse architecture is that is was designed to operate as a fully self-contained subsystem within the NF9000. Unlike basic system monitoring software that is typically available with Pentium Pro servers, IntraPulse has its own processors, software, internal network and power system. Where a system monitoring application that runs on the host operating system will simply quit when that server begins to experience problems or fails, IntraPulse will continue to operate and provide the system administrator with critical system information, regardless of the operational status of the server.

The NF9000 contains nine dedicated IntraPulse processors, each responsible for monitoring one of the system's primary subsystems. IntraPulse monitors the NF9000's main processor board, the system interface, the backplane chassis controller, the DIIM™ (Dynamic I/O Isolation Module) controller, and the remote interface card. Additional, IntraPulse maintains a system recorder, which records all system traffic in a circular NVRAM (Non-Volatile RAM) buffer. With real-time time and date referencing, the system recorder enables system administrators to re-construct system activity by accessing the log.

The IntraPulse system can be managed either locally, through a dial-up connection or through the enterprise network. The information collected and analyzed by IntraPulse can be presented to a system administrator either through NetFRAME's Maestro system management software, or through a local or dial-in terminal.

The following provisional patent applications, commonly owned and filed on the same day as the present application, are related to the present application and are incorporated by reference:

COMPUTER SYSTEM HARDWARE INFRASTRUCTURE FOR HOT PLUGGING MULTI-FUNCTION PCI CARDS WITH EMBEDDED BRIDGES (6391-704); invented by:

Don Agneta
Stephen E.J. Papa
Michael Henderson
Dennis H. Smith
Carlton G. Amdahl
Walter A. Wallach

COMPUTER SYSTEM HARDWARE INFRASTRUCTURE FOR HOT PLUGGING SINGLE AND MULTI-FUNCTION PC CARDS WITHOUT EMBEDDED BRIDGES (6391-705); invented by:

Don Agneta
Stephen E.J. Papa
Michael Henderson
Dennis H. Smith
Carlton G. Amdahl
Walter A. Wallach

ISOLATED INTERRUPT STRUCTURE FOR INPUT/OUTPUT ARCHITECTURE (6391-706); invented by:

Dennis H. Smith
Stephen E.J. Papa

THREE BUS SERVER ARCHITECTURE WITH A LEGACY PCI BUS AND MIRRORED I/O PCI BUSES (6391-707); invented by:

Dennis H. Smith
Carlton G. Amdahl
Don Agneta

HOT PLUG SOFTWARE ARCHITECTURE FOR OFF THE SHELF OPERATING SYSTEMS (6391-708); invented by:

Walter A. Wallach
Mehrdad Khalili
Mallikarunan Mahalingam
John Reed

REMOTE SOFTWARE FOR MONITORING AND MANAGING ENVIRONMENTAL MANAGEMENT SYSTEM (6391-709); invented by:

Ahmad Nouri

REMOTE ACCESS AND CONTROL OF ENVIRONMENTAL MANAGEMENT SYSTEM (6391-710); invented by:

Karl Johnson
Tahir Sheik

HIGH PERFORMANCE NETWORK SERVER SYSTEM MANAGEMENT INTERFACE (6391-711); invented by:

Srikumar Chari
Kenneth Bright
Bruno Sartirana

CLUSTERING OF COMPUTER SYSTEMS USING UNIFORM OBJECT NAMING AND DISTRIBUTED SOFTWARE FOR LOCATING OBJECTS (6391-712); invented by:

Walter A. Wallach
Bruce Findley

MEANS FOR ALLOWING TWO OR MORE NETWORK INTERFACE CONTROLLER CARDS TO APPEAR AS ONE CARD TO AN OPERATING SYSTEM (6391-713); invented by:

Walter A. Wallach
Mallikarunan Mahalingam

HARWARE AND SOFTWARE ARCHITECTURE FOR INTER-CONNECTING AN ENVIRONMENTAL MANAGEMENT SYSTEM WITH A REMOTE INTERFACE (6391-714); invented by:

Karl Johnson
Walter A. Wallach
Dennis H. Smith
Carl G. Amdahl

SELF MANAGEMENT PROTOCOL FOR A FLY-BY-WIRE SERVICE PROCESSOR (6391-715); invented by:

Karl Johnson
Walter A. Wallach
Dennis H. Smith
Carl G. Amdahl

NetFRAME®

# Wire Service Implementation

## User's Guide

Version 1.0

February 28, 1997

Prepared for:

NetFRAME Wire Service Implementation Group

by:

Gary Liu

Ken Nguyen

Trademarks:

# Table of Contents

# 1. Preface

## 1.1 How this Manual is Organized

This manual consists of the following:

| Chapter | Description |
|---|---|
| 2. Introduction | Introduces the Wire System subsystem of Raptor. |
| 3. Command Protocol | Describes the generic command format for the Wire Service command protocol. It also describes the data types used in this protocol. |
| 4. System Bus Interface | Describes the System Bus Interface. |
| 5. Special Considerations | Describes some special considerations for using switches, the power up process, some monitoring operations, LEDs, and critical events. |
| 6. Remote Interface Serial Protocol | Describes the protocol used to transfer Wire Service messages to a remote management processor. |
| 7. Callout Script Syntax | Describes the callout syntax when the Remote Interface is requested to make a callout. |
| 8. Include File Information | Describes the System Definition Language file. |
| 9. Event ID Codes | Describes the event ID codes. |
| 10. Glossary | Describes the terms used in Wire Service Implementation. |

## 1.2 Audience

This manual is written for personnel who implement low-level drivers and SNMP agents. It is not written for end users or Marketing.

## 1.3 Documentation

The NetFRAME *Wire Service Implementation Command Reference* is a companion manual. You can use that manual to learn about the commands.

A limited knowledge of $I^2C$ protocol is a prerequisite to understanding Wire Service Protocols. Refer to *The $I^2C$-bus and How to Use It*, (Philips Semiconductor, January 1992) for more information on $I^2C$ protocol. (Ken Nguyen has a copy of this document.)

This manual implements the Wire Service architecture. Refer to *Raptor Wire Service Architecture, Version 1.3*, (NetFRAME, October 1996) for details on this architecture. It contains a section that describes the physical signal connections to the Wire Service processors. It also contains detailed diagrams of the Wire Service System, and Wire Service Interface Programming State Diagram.

do I need this?

## 1.4 Acknowledgment

Kent Tsai prepared 4.System Bus Interface.

## 1.5 Summary f Amendments

| | | |
|---|---|---|
| Version 1.0 | Initial Version | December 17, 1996 |
| | Final Version | February 28, 1997 |

## 2. Introduction

The Wire Service subsystem is a c mponent f the Raptor System. The Wire Service subsystem is hereafter referred t as Wire Service. It is a network of micro controllers, and its purpos is to transfer messages to the other components of the Raptor System. Wire Service is comprised of system c ntrol, diagnostic, maintenance, and logging processors. The Wire Service processors on the System Board are interconnected with an ISA bus, and processors on the Back Plane are interconnected with an $I^2C$ serial bus.

One of the processors on the System Board is called the System Interface processor. It is part of the System Bus Interface (SBI), which is the interface between the Wire Service processors and the CPUs on the System Board. The SBI is an integral component of Wire Service. Messages are transferred back and forth among the CPUs on the ISA bus and the Wire Service processors on the $I^2C$ bus, through this interface.

Wire Service is entirely a "fly-by-wire" system. There are no switches, indicators, or other controls, which are directly connected to the function it monitors or controls. All of the monitor and control connections are made by the various Wire Service processors. These processors are Microchip PIC processors and the Back Plane of Wire Service is a 400 kbps $I^2C$ serial bus. Control on this bus is distributed. Each processor can be a sender (a master) or a receiver (a slave) and each is interconnected by this bus. A processor directly controls its own resources, and indirectly controls resources of other processors on the bus. An overview of Wire Service is illustrated in Figure 1. Wire Service Overview, on page 2.

The Wire Service processors are shown in the boxes that have thick lines. Each box includes the processor name, such as System Recorder, which is frequently referred to as the black box, and a unique processor id (PI), such as 01. The $I^2C$ bus is shown with a thick line, while the ISA bus is shown with a thin line.

The System Recorder and the Chassis Control are the first micro c ntr llers to power up. The canisters are not shown as being n the Back Plane because they are removable. In addition, the Remote Interface is also removable.

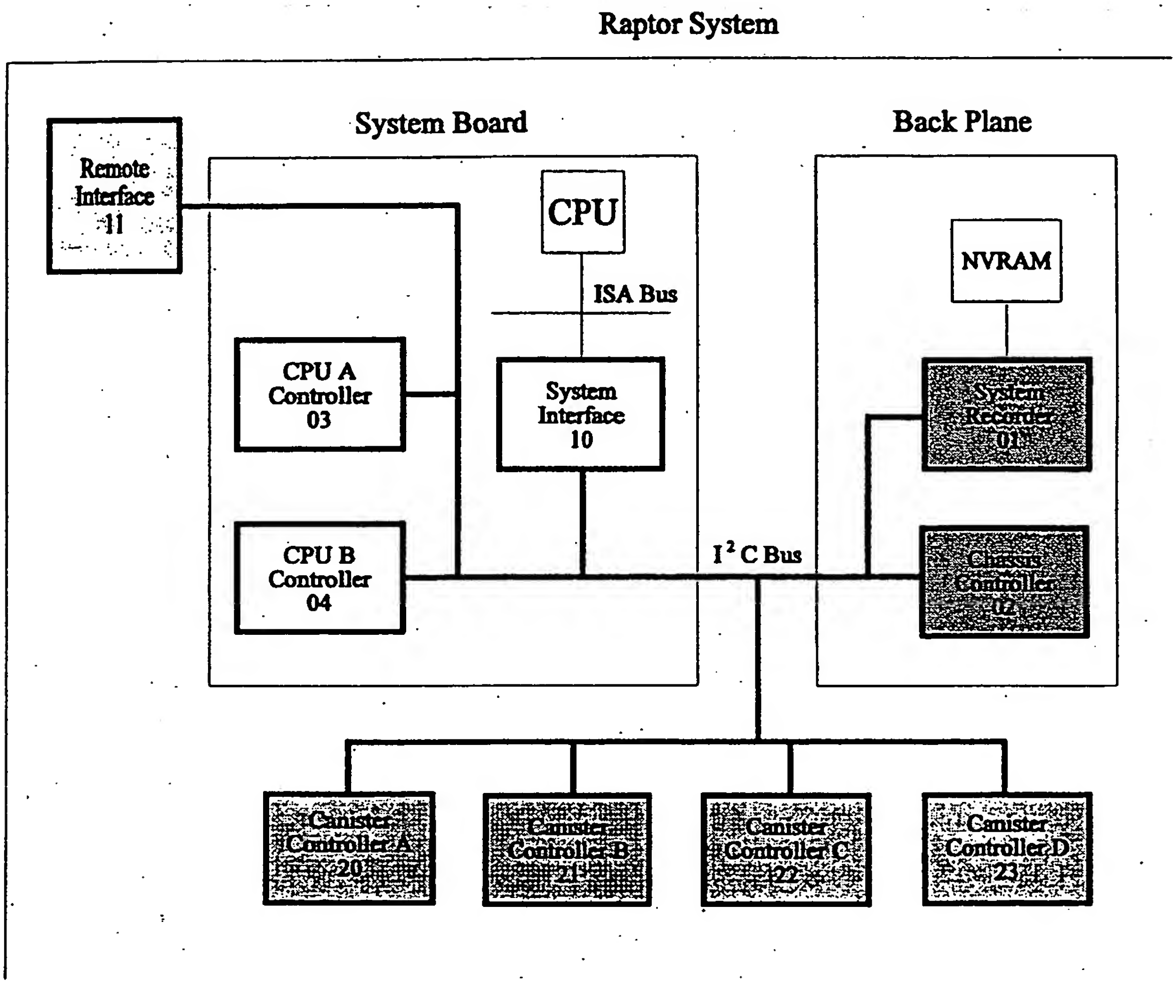


*Figure 1. Wire Service Overview*

## 3. Command Protocol

The command, diagnostic, monitoring, and logging functions of Wire Service are accessed through the common I$^2$C bus protocol. The processors that comprise Wire Service utilize this bus f r communicati n. They are interconnected by this bus.

The I$^2$C bus protocol uses an address in Wire Service memory as the means of identifying the various commands. Any function can be queried by generating a "read" request, which has its address as part of its protocol format. Conversely, a function can be executed by "writing" to an address specified in the protocol format. Any Wire Service processor can initiate read and write requests by sending a message on the I$^2$C bus to the processor responsible for that function.

Multiple simultaneous read requests can result in errors, particularly read requests to the log. Several sequence numbers may be lost in a string of messages. You should use I$^2$C bus contention to ensure that multiple read requests are not simultaneous.

This protocol includes data types as part of the address data. It implements a separate address space for each data type. Refer to Section 3.2 Data Types on page 7, for more information on data types. The ability to use separate address spaces allows for compact internal storage. It also allows you to create unique and more complex data types, which are better suited to specific functions.

### 3.1 Generic Format for Command Protocol

A generic protocol format is used for the Wire Service commands. The formats are shown as tables for read requests and responses, and write requests and responses. These tables are shown in Section 3.1.1 Protocol Tables, on page 3. The fields in this protocol are described in Section 3.1.2 Protocol Field Descriptions, on page 5.

Both read and write requests and responses must be initiated. This is indicated by a Master Asserts START. For example, an external processor wants to set the temperature of all sensors. This is initiated with the WS_SYS_TEMP_DATA command. The master writes this command to the slave. For this specific transaction, the slave receives command data from the master. This data is transmitted to the slave, byte by byte. That is, byte 0 is transmitted, then byte 1, and so on, until byte N+5 has been transmitted. The slave then transmits the requested data back to the master. The Master Repeats START and receives (reads) the data.

#### 3.1.1 Protocol Tables

A generic read request and response table, and a generic write request and response table are shown in Table 1. Generic Format Protocol on page 4. The byte fields that are grey indicate that the contents of the byte were calculated in the Wire Service firmware.

A Wire Service read and write request consists of a payload, a message, and a packet. Payload is the data included in the request. Referring to Table 1, payload for a read request is byte 4, and for a read response, it is byte 1 through byte N+1. Payload for a write request is byte 4 through byte N+4, and for a write response it is byte 1. Message is a wrapper around this data. In addition to the data, it includes **Slave Addr, LSBit, MSBit, Type, Command ID (LSB and MSB)**, and **Status**. Packet is a wrapper around a message that is transferred to the ISA bus. It includes **Check Sum** and the **Inverted Slave Addr** fields.

*Table 1. Generic Format Protocol*

## READ

Master Asserts START (Request)

| Offset | | |
|---|---|---|
| Byte 0 | Slave Addr (7 bits) | 0 LSBit |
| Byte 1 | MSBit (1) | Type |
| Byte 2 | Command ID (LSB) | |
| Byte 3 | Command ID (MSB) | |
| Byte 4 | Read Request Length (N) | |
| Byte 5 | Check Sum | |

Master Repeats START (Response)

| Offset | | |
|---|---|---|
| Byte 0 | Slave Addr (7 bits) | 1 LSBit |
| Byte 1 | Read Response Length (N) | |
| Byte 2 | Data Byte 1 | |
| : | : | |
| Byte N+1 | Data Byte N | |
| Byte N+2 | Status | |
| Byte N+3 | Check Sum | |
| Byte N+4 | Inverted Slave Addr | |

## WRITE

Master Asserts START (Request)

| Offset | | |
|---|---|---|
| Byte 0 | Slave Addr (7 bits) | 0 LSBit |
| Byte 1 | MSBit (0) | Type |
| Byte 2 | Command ID (LSB) | |
| Byte 3 | Command ID (MSB) | |
| Byte 4 | Write Request Length (N) | |
| Byte 5 | Data Byte 1 | |
| : | : | |
| Byte N+4 | Data Byte N | |
| Byte N+5 | Check Sum | |

Master Repeats START (Response)

| Offset | | |
|---|---|---|
| Byte 0 | Slave Addr (7 bits) | 1 LSBit |
| Byte 1 | Write Response Length (0) | |
| Byte 2 | Status | |
| Byte 3 | Check Sum | |
| Byte 4 | Inverted Slave Addr | |

### 3.1.2 Protoc l Field Descriptions

The protocol fields are described in the following table. These fields can be modified only by Wire Service firmware.

| FIELD | DESCRIPTION |
|---|---|
| Slave Addr | Specifies the processor identification code. This field is 7 bits wide. Bit [7...1]. |
| LSBit | Specifies what type of activity is taking place. If LSBit is clear (0), the master is writing to a slave. If LSBit is set (1), the master is reading from a slave. |
| MSBit | Specifies the type of command. It is bit 7 of byte 1 of a request. If this bit is clear (0), this is a write command. If it is set (1), this is a read command. |
| Type | Specifies the data type of this command, such as bit or string. |
| Command ID (LSB) | Specifies the least significant byte of the address of the processor. |
| Command ID (MSB) | Specifies the most significant byte of the address of the processor. |
| Length (N) | |
| Read Request | Specifies the length of the data that the master expects to get back from a read response. The length, which is in bytes, does not include the **Status**, **Check Sum**, and **Inverted Slave Addr** fields. |
| Read Response | Specifies the length of the data immediately following this byte, that is byte 2 through byte N+1. The length, which is in bytes, does not include the **Status**, **Check Sum**, and **Inverted Slave Addr** fields. |
| Write Request | Specifies the length of the data immediately following this byte, that is byte 2 through byte N+1. The length, which is in bytes, does not include the **Status**, **Check Sum**, and **Inverted Slave Addr** fields. |
| Write Response | Always specified as 0. |
| Data Byte 1<br>:<br>Data Byte N | Specifies the data in a read request and response, and a write request. |
| Status | Specifies whether or not this command executes successfully. A non-zero entry indicates a failure. Status codes are described in. Section 3.3 Wire Service Status Codes on page 15. |
| Check Sum | Specifies a direction control byte to ensure the integrity of a message on the wire. This byte is calculated in the Wire Service firmware. |
| Inverted Slave Addr | Specifies the Slave Addr, which is inverted. This byte is calculated in the Wire Service firmware. |

### 3.1.3 W rking with the Fields

The values used in the protocol are derived from the address of a specific processor. This address consists of three parts:

1. Processor identification code
2. Data type
3. Subaddress

The address is 4 bytes in length and is in hexadecimal notation, as follows:

**PIDTALAMh**

Where:

| | | |
|---|---|---|
| **PI** | is | Processor ID |
| **DT** | is | Data type |
| **AL** | is | First byte of the 2-byte subaddress, that is **Command ID**. It is the least significant byte, or LSB. |
| **AM** | is | Second byte of the 2-byte subaddress, that is **Command ID**. It is the most significant byte, or MSB. |
| **h** | is | Hexadecimal notation. |

#### *3.1.3.1 Slave Addr*

The identification codes for the processors are listed in the following table. The processor identification (PI) is used to determine the **Slave Addr**.

To determine the **Slave Addr** value for a request, shift all the bits of the **PI** to the left one bit. For example, the processor ID for the chassis controller is 02h, so after the bits are shifted, the **Slave Addr** is 04h. To determine the **Slave Addr** value for a response, increment the shifted value by one (1). In this case, the response **Slave Addr** is 05h.

| Processor Name | PI |
|---|---|
| System Recorder | 01h |
| Chassis Controller | 02h |
| CPU A Controller | 03h |
| CPU B Controller | 04h |
| System Interface | 10h |
| Remote Interface | 11h |
| Canister Controller A | 20h |
| Canister Controller B | 21h |
| Canister Controller C | 22h |
| Canister Controller D | 23h |

#### *3.1.3.2 Type*

Valid data types (DT) for the commands are listed bel w. They determine the value specified in the Type field. The value for a write request is the value specified in DT. The value for a read request is the value specified in DT, with bit 7 set.

| DATA TYPE | READ | WRITE |
|---|---|---|
| Bit | 81h | 01h |
| Byte | 82h | 02h |
| String | 83h | 03h |
| Log | 84h | 04h |
| Event | 85h | 05h |
| Queue | 86h | 06h |
| Byte Array | 87h | 07h |
| Lock | 88h | 08h |
| Screen | 89h | 09h |

#### *3.1.3.3 Command ID*

The subaddress (ALAM) is used to determine the **Command ID**. The first byte, or **AL**, is the least significant byte, or LSB. The second byte, or **AM**, is the most significant byte, or MSB.

## 3.2 Data Types

All of the data types used in Wire Service commands are described in this section. Each data type includes a description and an example of a simple protocol command. The format of the examples is the same for all of the commands.

**Check Sum** and **Inverted Slave Addr** are not shown in the examples for simplicity. These bytes are calculated in the firmware. **Check Sum** is appended and checked by all commands when they are physically sent on the Wire Service bus.

### 3.2.1 Bit Data Type

You can use the bit data type for a simple logic value, such as TRUE (1) and FALSE (0), or ON (1) and OFF (0).

**Example:**

Read Bit Message:

Request

| | | | | |
|---|---|---|---|---|
| Slave Addr | Bit Type<br>Read | Command ID<br>(LSB) | Command ID<br>(MSB) | Request Length<br>1 |

Response

| | | | |
|---|---|---|---|
| Slave Addr | Length<br>1 | Bit Value<br>0/1 | Status<br>0 |

Write Bit Message:

Request

| | | | | | |
|---|---|---|---|---|---|
| Slave Addr | Bit Type<br>Write | Command ID<br>(LSB) | Command ID<br>(MSB) | Length<br>1 | Bit Value<br>0/1 |

Response:

| | | |
|---|---|---|
| Slave Addr | Length<br>0 | Status<br>0 |

### 3.2.2 Byte Data Type

You can use the byte data type for a single-byte value, with a variable length of 0 through FF.

**Example:**

Read Byte Message:

Request

| | | | | |
|---|---|---|---|---|
| Slave Addr | Byte Type<br>Read | Command ID<br>(LSB) | Command ID<br>(MSB) | Request Length<br>1 |

Response

| | | | |
|---|---|---|---|
| Slave Addr | Length<br>1 | Byte Value | Status<br>0 |

Write Byte Message:

Request

| | | | | | |
|---|---|---|---|---|---|
| Slave Addr | Byte Type<br>Write | Command ID<br>(LSB) | Command ID<br>(MSB) | Length<br>1 | Byte Value |

Response:

| | | |
|---|---|---|
| Slave Addr | Length<br>0 | Status<br>0 |

### 3.2.3 String Data Type

You can use the string data type for a variable-length string of data of 0 to FF bytes. The maximum number of bytes allowed is FF. Exceeding this maximum causes an error condition.

**Example:**

<u>Read String Message:</u>

Request

| | | | | |
|---|---|---|---|---|
| Slave Addr | String Type<br>Read | Command ID<br>(LSB) | Command ID<br>(MSB) | Request Length<br>N |

Response

| | | | | | |
|---|---|---|---|---|---|
| Slave Addr | Length<br>N | String Data<br>1 | ... | String Data<br>N | Status<br>0 |

<u>Write String Message:</u>

Request

| | | | | | |
|---|---|---|---|---|---|
| Slave Addr | String Type<br>Write | Command ID<br>(LSB) | Command ID<br>(MSB) | Length<br>N | ... |

| | | |
|---|---|---|
| String Data<br>1 | ... | String Data<br>N |

Response:

| | | |
|---|---|---|
| Slave Addr | Length<br>0 | Status<br>0 |

### 3.2.4 Lock Byte Data Type

(NOT IMPLEMENTED AT THIS TIME)

### 3.2.5 Byte Array Data Type

You can use the byte array data type for general storage of data that is not anticipated in the implementation of Wire Storage. This data type has a length of 0 to FF bytes. This storage is implemented with NVRAM, so external code manages the allocation and deallocation of data directory information.

**Example:**

Read Byte Array Message:

Request

| Slave Addr | Byte Array Type | Command ID | Command ID | Request Length |
|---|---|---|---|---|
| | Read | (LSB) | (MSB) | N |

Response

| Slave Addr | Length | Array Data | ... | Array Data | Status |
|---|---|---|---|---|---|
| | N | 1 | | N | 0 |

Write Byte Array Message:

Request

| Slave Addr | Byte Array Type | Command ID | Command ID |
|---|---|---|---|
| | Write | (LSB) | (MSB) |

...

| Length | Array Data | ... | Array Data |
|---|---|---|---|
| N | 1 | | N |

Response:

| Slave Addr | Length | Status |
|---|---|---|
| | 0 | 0 |

### 3.2.6 Log Data Type

You can use the log data type to write a byte string to a circular log buffer. This data type records system events in the NVRAM system log. The maximum number of bytes that can be written in a log entry is 249. The log processor adds 6 bytes of ID and a timestamp at the beginning of the command. Refer to Section 5.6 Timestamp Considerations on page 31, for more information on the timestamp.

The addressing of log entries has the following special considerations:

- Address 65534 (FFFEh) specifies the address of the oldest valid message in a read operation.
- Address 65533 (FFFDh) specifies the address of the next message in sequence from the last message read from the log.

- The address of log messages wraps at 65279 (FEFFh). The next sequential message after this address is 0.
- Only the 65534 and 65533 addresses are recognized for a read operation. These addresses are ignored for a write operation and the next available sequential message is written.
- To read the entire log in forward-time order, read the timestamp first. Then, read the message at address 65534. This is the first message. Then, read the message at address 65533 to get the next sequential message. Repeat this last step until the **Status** field returns a non-zero value, which indicates a failure.
- If you want to keep a complete external copy of the log, read the entire log in forward-time order. Then, periodically read from the next valid message at address 65533 to the end, and add that to the external copy.

Multiple simultaneous read requests to the log can result in errors. Several sequence numbers may be lost in a string of messages. Use $I^2C$ bus contention to ensure that multiple read requests are not simultaneous.

Log messages may be out of sequence if the log is filled so quickly that old messages are written over before they are read. When this occurs, the log processor returns the oldest currently available entry for the next entry read.

**Example:**

Read Log Message:

Request

| | | | | |
|---|---|---|---|---|
| Slave Addr | Log Type<br>Read | Log Addr<br>(LSB) | Log Addr<br>(MSB) | Request Length<br>FF |

Response

| | | | | | | |
|---|---|---|---|---|---|---|
| Slave Addr | Length<br>10 | Command ID<br>(LSB) | Command ID<br>(MSB) | Log Time<br>(LSB) | Log Time | ... |

| | | | | | |
|---|---|---|---|---|---|
| Log Time | Log Time<br>(MSB) | Log Data<br>Byte 1 | ... | Log Data<br>Byte 4 | Status<br>0 |

Write Log Message:

Request

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Slave Addr | Log Type<br>Write | 00 | 00 | Length<br>4 | Log Data<br>Byte 1 | ... | Log Data<br>Byte 4 |

Response:

| | | |
|---|---|---|
| Slave Addr | Length<br>0 | Status<br>0 |

All events are written to the log as a 4-byte message, as shown below:

| Byte | Description |
|---|---|
| Byte 1 | Severity level |
| Byte 2 | Source and encoding byte |
| Byte 3 | Event ID LSB |
| Byte 4 | Event ID MSB |

Conventions for the log data portion of the message are listed below. These conventions are used by processors external to Wire Service, and are used in conjunction with $I^2C$ bus contention.

**Log Data Byte 1:** Severity Levels Byte are listed below:

| | |
|---|---|
| 00h | Unknown |
| 10h | Informational |
| 20h | Warning |
| 30h | Recoverable error |
| 40h | Unrecoverable or Fatal Error |

**Log Data Byte 2:** Source and Encoding Byte. The processor that logged the message in the four high bits of the byte are listed below:

| | |
|---|---|
| 00h | Wire Service Internal |
| 10h | Onboard Diagnostics |
| 20h | External Diagnostics |
| 30h | BIOS |
| 40h | Time Synchronizer |
| 50h | Windows |
| 60h | Windows/NT |
| 70h | NetWare |
| 80h | OS/2 |
| 90h | UNIX |
| A0h | VAX/VMS |

The processor that logged the entry in the four low bits of the byte are listed below:

| | |
|---|---|
| 00h | Binary |
| 10h | ASCII |
| 20h | Unicode |

For example, an external diagnostics ASCII error message has a severity byte value of 30h and a source and encoding byte value of 21h.

Additional conventions apply to binary encoded messages. Bytes 3 and 4 of the log data portion of the message are used as the LSB and MSB of a 16-bit message identifier. Each processor that logs binary messages, such as OS or BIOS, must maintain a file using the same format that contains the definition of all possible binary messages from their source. This file contains the identifier value, formatting string, and descriptive comments for each message. The processor must maintain the message identifier and supply it in a file to any requester. Log bytes 5 and higher of the log data portion of the message are message arguments in format-list order. Keep in mind that obsolete message identifiers cannot be reused because of having to continue to support older versions (backward compatibility).

### 3.2.7 Event Data Type

You can use the event data type to alert external interfaces of events in Wire Service. Event memory is organized as a bit vector, which has a minimum of 16 bits. Each bit in the bit vector represents a particular type of event. Writing an event sets the bit that represents the event in the bit vector.

Reading the event data type returns one or more event IDs, depending on the request length and events actually pending in the interface. Once an event is read, the corresponding event bit is cleared.

Valid event types are:

CPU status change

Power status change

Canister status change

Fan status change

Temperature

Screen (not implemented)

Queue (not implemented)

OS Timeout

**Example:**

Read Event Message:

Request

| | | | | |
|---|---|---|---|---|
| Slave Addr | Event Type Read | 01 | 00 | Request Length 10 |

Response

| | | | | | |
|---|---|---|---|---|---|
| Slave Addr | Length<br>N(1-16) | Event ID | ... | Event ID | Status<br>0 |

Write Event Message:

Request

| | | | | | |
|---|---|---|---|---|---|
| Slave Addr | Event Type<br>Write | 01 | 00 | Length<br>1 | Event ID<br>0-F |

Response:

| | | |
|---|---|---|
| Slave Addr | Length<br>0 | Status<br>0 |

### 3.2.8 Screen Data Type

You can use the screen data type to communicate character mode screen data from BIOS to the Remote Interface processor. This data can be from 0 to FF bytes in length.

The screen address space consists of an image of character video memory for an 80x50 screen. This 8000 byte block of memory represents the contents of the screen. There are 50 rows, and each row consists of 80 characters. Each character consists of two bytes. This represents a total of 4000 character pairs.

Each character cell has both a character byte and an attribute byte. Screen memory consists of 8191 bytes. However, memory above the address of 8000 is reserved for BIOS and remote management software. It is recommended that address 8001 and address 8002 be used as the cursor address register.

The Command ID (LSB) and Command ID (MSB) serve as an offset to get to a given location in this screen address space.

**Example:**

Read Screen Data Message:

Request

| | | | | |
|---|---|---|---|---|
| Slave Addr | Screen Data Type<br>Read | Command ID<br>(LSB) | Command ID<br>(MSB) | Request Length<br>N |

Response

| | | | | | |
|---|---|---|---|---|---|
| Slave Addr | Length<br>N | Screen Data<br>1 | ... | Screen Data<br>N | Status<br>0 |

Write Screen Data Message:

Request

| | | | | |
|---|---|---|---|---|
| Slave Addr | Screen Data Type<br>Write | Command ID<br>(LSB) | Command ID<br>(MSB) | Length<br>N |

...

| | | |
|---|---|---|
| Screen Data<br>1 | ... | Screen Data<br>N |

Response:

| | | |
|---|---|---|
| Slave Addr | Length<br>0 | Status<br>0 |

### 3.2.9 Queue Data Type

(NOT IMPLEMENTED AT THIS TIME)

## 3.3 Wire Service Status Codes

If the Wire Service **Status** byte contains a non-zero number, an error has occurred. The error codes are described as follows:

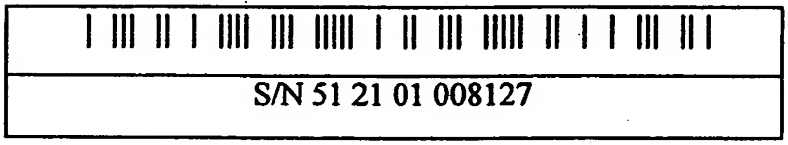
| CODE | DESCRIPTION |
|---|---|
| 1 | The slave did not respond to a request initiated by a master. Resubmit the command with a correct **Slave Addr**. |
| 2 | The slave did not have the data type or address specified in a request sent by the master. Resubmit the command with the correct data **Type** or **Command ID**. |
| 3 | The message or response is not valid. The **Slave Addr** is invalid. The master does not recognize the response. No further messages are sent for this transaction. |
| 4 | The message could not be completely sent or received. The master does not recognize the response. No further messages are sent for this transaction. |
| 5 | Message data in the **Check Sum** field was received incorrectly. The request should be resent, if possible. |
| 6 | The slave operation is not valid, such as a write request to a read-only command. |
| 7 | The slave responded that there was no data at the specified address for a queue or a log. |

## 3.4 Storing Serial Numbers

Serial numbers are stored in the DS2502 chip. Decimal data is stored in BCD format. For example, a hex dump of the EPROM bytes for sequential number 123456 is "123456", and a hex dump of Revision 87 is "87". There is a total of 16 bytes per serial number slot. Refer to NetFRAME Policies and Procedures *Serial Numbers*, Part Number 33-100053-01, for detailed information on serial numbers.

| Field | Name | Type | Bytes | Notes |
|---|---|---|---|---|
| RR | Revision | Decimal | 1 | |
| CC | Category | Hex | 1 | |
| TT | Type | Hex | 1 | |
| SSSSSS | Sequence Number | Decimal | 3 | Stored as 2 digits per byte. |
| FF | Factory | Hex | 1 | |
| MMDDYY | Date Code | Decimal | 3 | 1 byte each for month, day, and year. |
| AAAAAA | Network Group Address | Hex | 3 | Stored as 2 digits per byte. |
| XXXXXX | (Not used) | | 3 | Not written (all binary 1s) |

The format of the bar-code is as follows:



Notes:

1. The size of the label is 2" x ¼".
2. The label is white with black text.
3. The label is made of Z-Ultimate 3000, which is UL/CSA recognized and has a service temperature range from -20° to +300° F.
4. The bar-code and text are approximately 0.090" high.
5. The printed serial number contains revision, category, type, and sequence number. The fields are separated by spaces.
6. The bar-code consists of the printed part of the serial number only. It does not include the "S/N" portion of the text or the spaces between the parts of the serial number.
7. The bar-code is Code 39.

Serial numbers are stored in the following commands in the System Recorder processor:

WS_SYS_BP_SERIAL

WS_SYS_CAN_SERIAL1

WS_SYS_CAN_SERIAL2

WS_SYS_CAN_SERIAL3

WS_SYS_CAN_SERIAL4

WS_SYS_RI_SERIAL

WS_SYS_SB_SERIAL

WS_SYS_PS_SERIAL1

WS_SYS_PS_SERIAL2

WS_SYS_PS_SERIAL3

## 4. System Bus Interface

The System Bus Interface (SBI) is the interface used by the client[1] to transfer Wire Service messages. The format of these request and response messages is described in Section 3.1 Generic Format for Command Protocol on page 3. Figure 2. System Bus Interface Block Diagram shows the components of SBI.

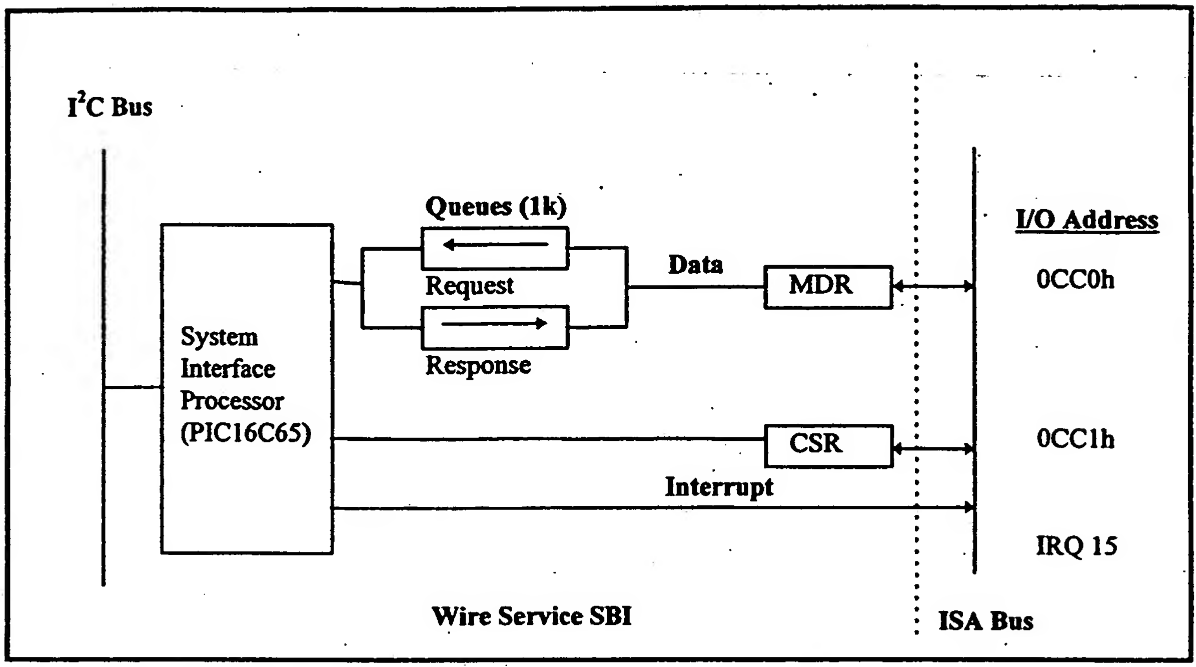


*Figure 2. System Bus Interface Block Diagram*

The SBI is located in system I/O storage as two registers Each register is 8 bits wide. One register is the Message Data Register (MDR) and the other is the Command and Status Register (CSR). The MDR transfers messages back and forth from Wire Service, and the CSR controls operations and reports on the status of commands. The MDR is located at 0CC0h and the CSR is located at 0CC1h. Both of these addresses are fixed.

Both synchronous and asynchronous I/O modes are provided by the SBI. An interrupt line provides the ability to request an interrupt when asynchronous I/O is complete, or when a Wire Service event occurs while the interrupt is enabled. This interrupt uses ISA IRQ 15, which is the interrupt number for only the ISA subsystem. This interrupt number address is also fixed.

### 4.1 Operation

The SBI is single-threaded. That is, only one client is allowed to access the SBI at a specific period of time. A program has to allocate the SBI for its use before using it, and then must deallocate it when an operation completes. The allocate fails if a previous allocation is not deallocated.

**Note:** It is very important to deallocate the SBI after each use.

In rare situations, more than one client may request allocation of the SBI at approximately the same time. Only one request is granted and the 'Interface Owner ID' field in the CSR indicates which client has allocated the SBI successfully. For more information refer to Section 4.2.2.1 CSR as a Status Register (Read) n page 20.

[1] Client is a software program that accesses the Wire Service through SBI, such as an OS driver or BIOS.

Send and receive Wire Service message, and Service Wire Service Event are two typical client perations. A sequence f commands are required to complete each peration. An operation usually begins with CSR status checking and SBI allocation. It is then followed by sending and receiving Wire Service messages, resetting CSR status and finally SBI reallocation. A suggested operational sequence is described in Section 4.4 Suggested Sequence f Operations on page 22.

## 4.2 SBI Components

The functions of the MDR and the SCR are described in this section.

### 4.2.1 MDR and Queues

The MDR and the request and response queues transfer Wire Service messages between a client and Wire Service. The queues utilize the first-in first-out (FIFO) technique. That is, the next message processed is the one that has been in the queue the longest time.

**Note:** These queues transfer one message at a time.

These queues have two functions:

1. They match speeds between the high-speed ISA bus and the slower Wire Service subsystem.
2. They serve as interim buffers for the transfer of messages. This relieves the System Interface processor of having to provide this buffer.

When the MDR is written from the ISA bus, it loads a byte into the request queue. When the MDR is read from the ISA bus, it unloads a byte from the response queue. The System Interface processor reads and executes the Wire Service request from the request queue when a message command is received in the CSR. A response message is written to the response queue when Wire Service completes executing the command. The client can read and write message data to and from the queues by executing a REP OUT/IN instruction through the MDR. Refer to *80x86 Programmer's Reference Manual* for more information on repeat input and output data to the I/O port.

### 4.2.2 CSR and Interrupts

The CSR has two functions. The first is to issue commands, and the second is to report on the status of execution of a command. The SBI commands are usually executed synchronously. That is, after issuing a SBI command, the client must keep polling the CSR status to confirm command completion. If the system issues a second command to the CSR before the System Interface processor reads the first one, a loss of data could result. The format of the CSR as a status register is described in Section 4.2.2.1 CSR as a Status Register on page 20.

In addition to synchronous I/O mode, the client can also request an asynchronous I/O mode for each SBI command by setting the 'Asyn Req' bit in the command. Wire Service requests an interrupt after the command executes. Only the 'Message' command is recommended for the asynchronous I/O mode. All other commands should use the synchronous I/O mode, because they either complete quickly or ignore the interrupt bit of the 'Reset' command.

The interrupt line uses ISA IRQ 15, which is not configurable under current implementation. Level-triggered (sensitive) is used for this interrupt line. A level-triggered interrupt request is recognized by keeping the signal at the same level. (An edge-triggered interrupt is recognized by the signal level transition).

The client can either enable r disable the level-triggered interrupt by sending 'Enable Ints' and 'Disable Ints' commands. If the interrupt line is enabled, the System Interface processor requests an interrupt, either when an asynchronous I/O is complete or when a Wire Service event occurs.

**Note:** Under current design, the Wire Service does not reset the interrupt. The client must issue a 'Clear Int Req' command t clear the interrupt after the interrupt request has been executed. Otherwise, it keeps receiving interrupt requests.

### *4.2.2.1 CSR as a Status Register (Read)*

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| Error | Int Ena | Events | Done | Int Req | Interface Owner ID | | |

Where:

| | |
|---|---|
| Interface Owner ID | 0: Interface not in use.<br>None-zero: Owner's ID for a successful allocation. |
| Int Req | 0: Synchronous I/O mode.<br>1: Asynchronous I/O mode. |
| Done | 0: Idle/command executing.<br>1: 'Message' command has completed. |
| Events | 0: No Wire Service event.<br>1: Wire Service events are pending. |
| Int Ena | 0: Interrupt is disabled.<br>1: Interrupt is enabled. |
| Error | 0: No error.<br>1: The last command written to the CSR was invalid. |

### *4.2.2.2 CSR as a Command Register (Write)*

The CSR command types and values are shown below. The System Interface processor keeps polling this register for a new command. When a new command is detected, it is executed immediately. However, if the same command is issued back-to-back, it is not recognized as a new command.

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| Asyn Req | Command Type | | | | | | |

Where:

| | |
|---|---|
| Asyn Req | 0: Request a synchronous I/O mode.<br>1: Request an asynchronous I/O mode. An interrupt request is generated when the c mmand has completed. |
| Command Type | Contains the command value that is executed by the System Interface processor. |

## 4.3 CSR Commands

The CSR commands are listed below. The field that confirms command completion is specified at the end of the description of each command.

| Command Type | Value | Description |
|---|---|---|
| Allocate | 1-7<br>0x01<br>:<br>0x07 | The first command in a sequence of commands. This command clears both data queues and the 'Done' bit. The same value is returned in the 'Interface Owner ID' field if the allocation is successful. A mismatch indicates that allocation has failed.<br>**Note:** The allocation always fails if the 'Interface Owner ID' is not zero before the 'Allocate' command is issued. |
| | | **Completion:** 'Interface Owner ID' is not set to zero. |
| Deallocate | 16<br>0x10 | The last command in a sequence of commands. This command clears the 'Done' bit and 'Interface Owner ID' field.<br>**Note:** You should never use this command to obtain access to the SBI, even though the design does not stop you from doing so. Using Deallocate in this manner results in unpredictable behavior. |
| | | **Completion:** The 'Done' and 'Interface Owner ID' bits are cleared. |
| Enable Ints | 17<br>0x11 | Enables interrupt for Wire Service events. This may cause an interrupt if the 'Events' bit is already set. |
| | | **Completion:** The 'Int Ena' bit is set. |
| Disable Ints | 18<br>0x12 | Disables interrupt for Wire Service events. |
| | | **Completion:** The 'Int Ena' bit is cleared. |
| Message | 19<br>0x13 | This command is issued after a Wire Service request message has been sent to the request queue through the MDR. The 'Done' bit is set after the Wire Service request message is processed and the response message is placed on the response queue. |
| | | **Completion:** The 'Done' bit is set. |

| | | |
|---|---|---|
| Clear Done | 32<br>0x20 | Clears the 'Done' bit and both queues. |
| | | **C mpletion:** The 'Done' bit is cleared. |
| Clear Int Req | 33<br>0x21 | Clears the 'Int Req' bit. It must be executed after receiving an interrupt in order to turn off the hardware interrupt request. |
| | | **Completion:** The 'Int Req' bit is cleared. |
| Reset | 90<br>0x5a | Unconditionally clears all bits in the CSR except the 'Events' bit. It aborts any currently in-progress message operation and clears any interrupt. |
| | | **Completion:** No need to wait. |
| Diagnostic Mode | A5 | Only used for firmware diagnostic purposes. |
| | | **Completion:** Not applicable. |
| Exit Diagnostic | 5A | Only used for firmware diagnostic purposes. |
| | | **Completion:** Not applicable. |

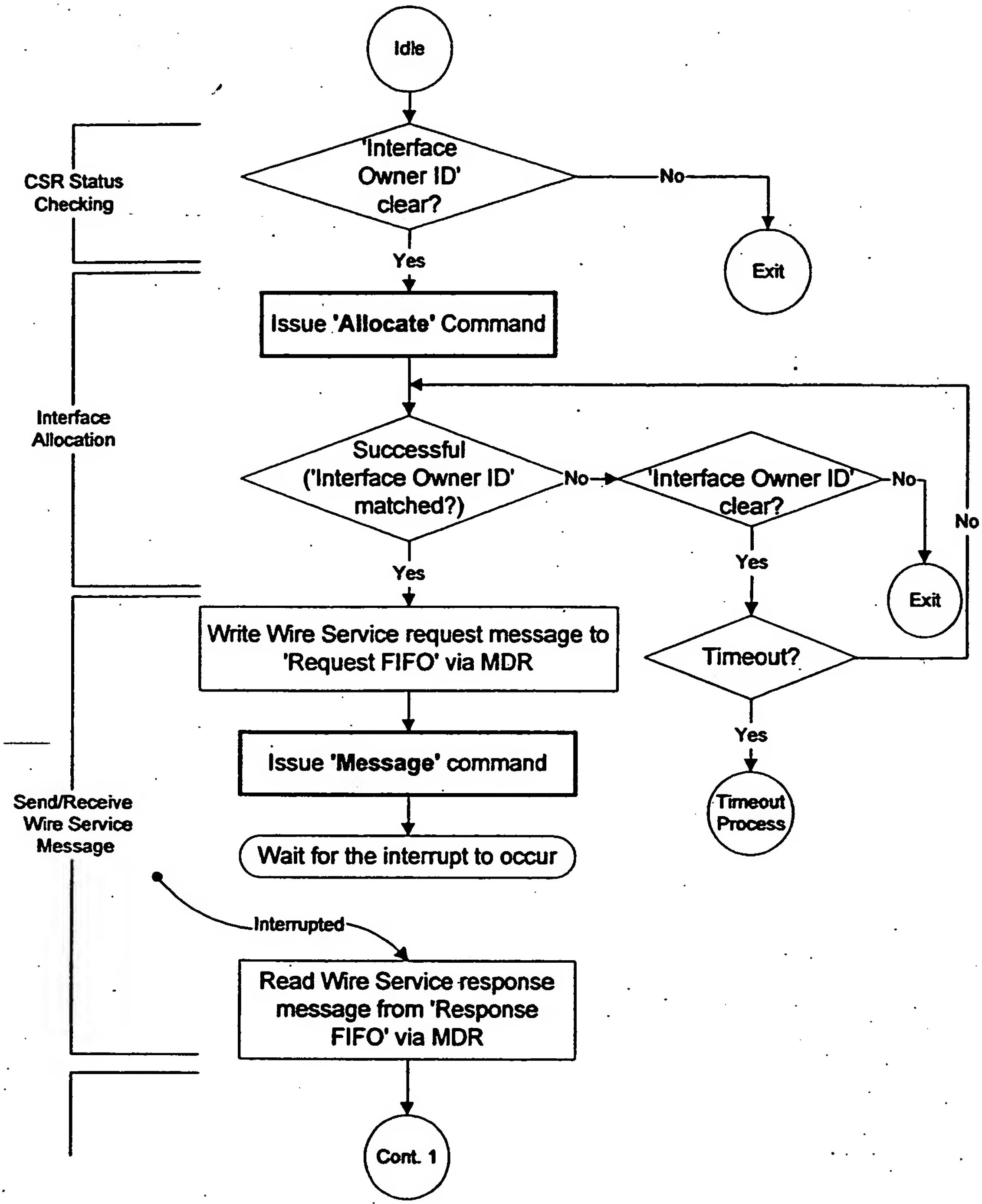
## 4.4 Suggested Sequence of Operations

Send and receive Wire Service message and Wire Service event are two typical operations. For each operation, a sequence of commands is issued, as shown in Section 4.4.1 Sending Requests and Receiving Responses on page 22, and Section 4.4.2 Wire Service Events on page 24. Because the completion of each command is usually confirmed by checking related CSR status, status and time-out checking are also reported.

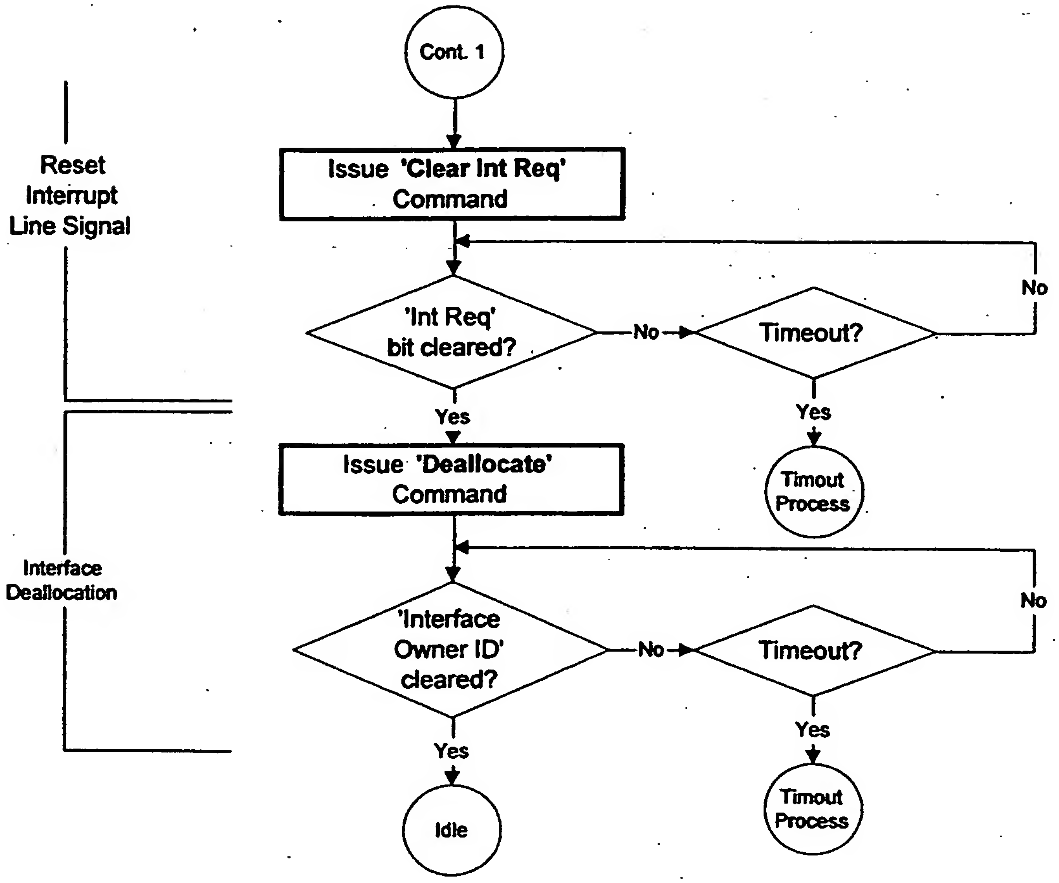
**Note:** It is strongly recommended that the client check related CSR status before issuing a command that relies on the CSR status as confirmation of command completion. In addition, since Wire Service is not implemented as a state machine, it is the responsibility of the client to ensure the right order of the sequence of commands. In other words, Wire Service neither keeps track of the states, nor does it reject the command that causes an invalid state transition.

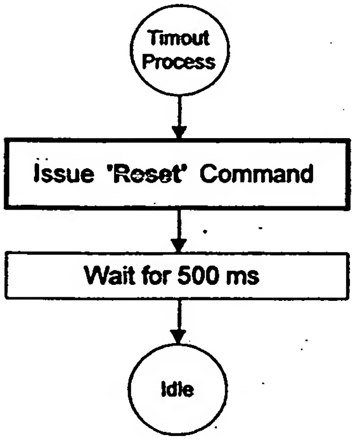
### 4.4.1 Sending Requests and Receiving Responses

This section describes the sequence of commands used for sending a Wire Service Request message and receiving its response message. In this sequence, all commands use synchronous I/O mode, except for the 'Message' command. When this I/O mode is used, wait for the 'Done' bit to be set instead of the interrupt. Timeouts should also be considered, as shown in Section 4.4.2 Wire Service Events on page 24.



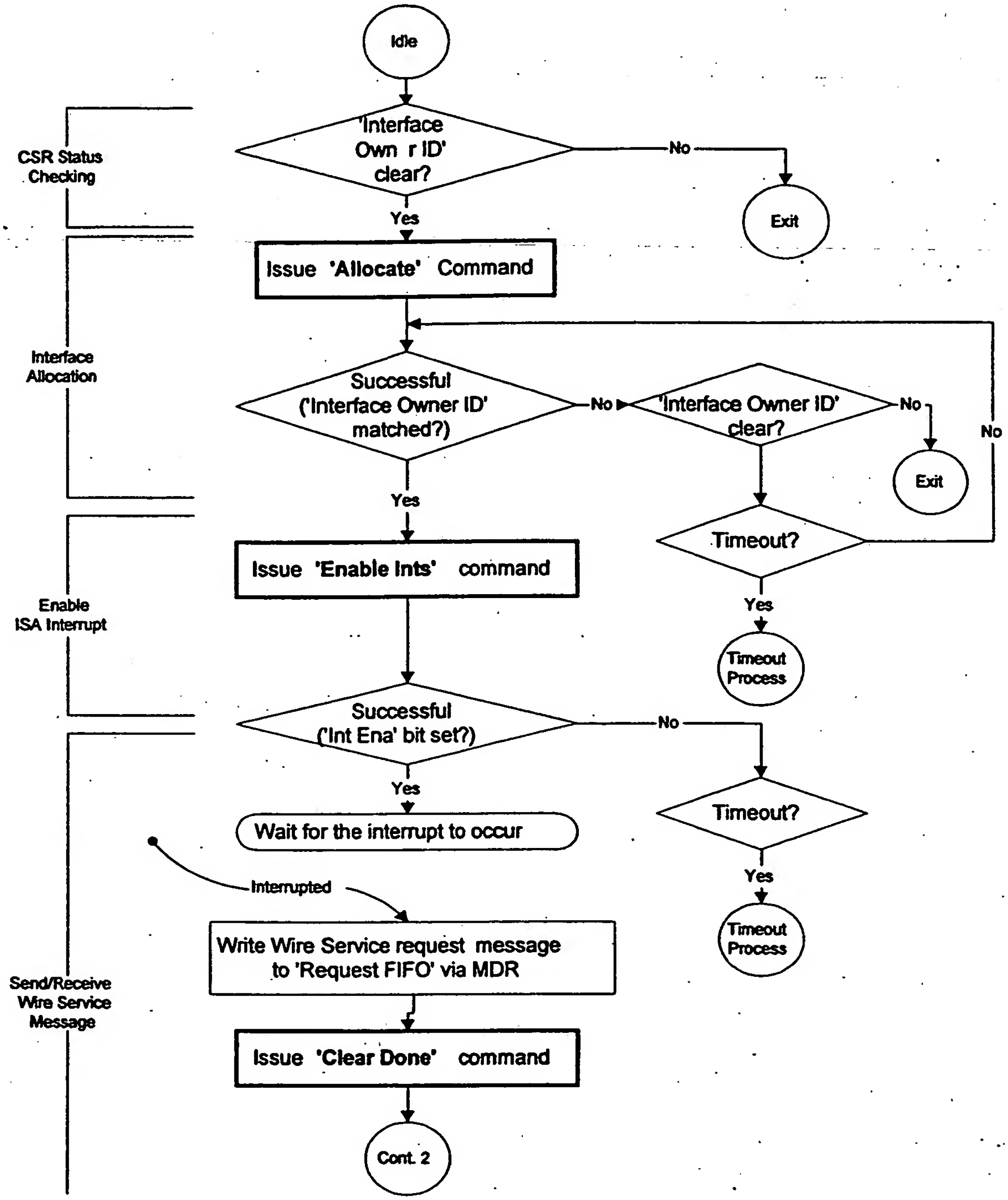
CSR Status Checking
Interface Allocation
Send/Receive Wire Service Message
Idle
'Interface Owner ID' clear?
No
Exit
Yes
Issue 'Allocate' Command
Successful ('Interface Owner ID' matched?)
No
'Interface Owner ID' clear?
No
Exit
No
Yes
Timeout?
Yes
Timeout Process
Yes
Write Wire Service request message to 'Request FIFO' via MDR
Issue 'Message' command
Wait for the interrupt to occur
Interrupted
Read Wire Service response message from 'Response FIFO' via MDR
Cont. 1

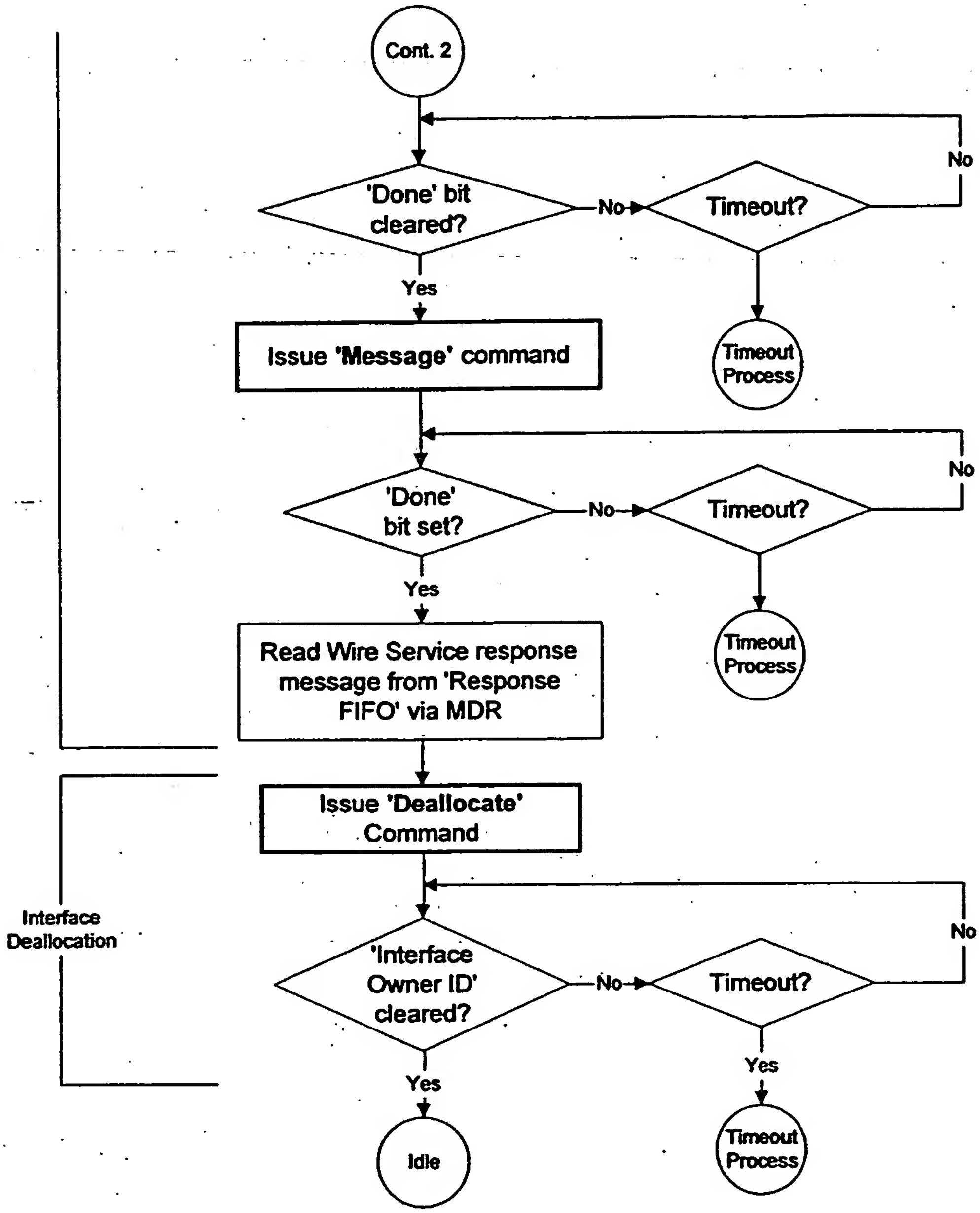




### 4.4.2 Wire Service Events

In this command sequence, the ISA bus interrupt is enabled to service a Wire Service event. Keep in mind that the client can also choose to poll the 'Events' bit until it is set. However, this is very inefficient. This command sequence is also used for a synchronous I/O mode 'Message' command.


Idle
'Interface Own r ID' clear?
No
Exit
Yes
CSR Status Checking
Issue 'Allocate' Command
Interface Allocation
Successful ('Interface Owner ID' matched?)
No
'Interface Owner ID' clear?
No
Exit
No
Yes
Timeout?
Issue 'Enable Ints' command
Yes
Timeout Process
Enable ISA Interrupt
Successful ('Int Ena' bit set?)
No
Yes
Timeout?
Wait for the interrupt to occur
Yes
Interrupted
Timeout Process
Write Wire Service request message to 'Request FIFO' via MDR
Send/Receive Wire Service Message
Issue 'Clear Done' command
Cont. 2


Cont. 2
'Done' bit cleared?
No
Timeout?
No
Yes
Issue 'Message' command
Timeout Process
'Done' bit set?
No
Timeout?
No
Yes
Read Wire Service response message from 'Response FIFO' via MDR
Timeout Process
Issue 'Deallocate' Command
Interface Deallocation
'Interface Owner ID' cleared?
No
Timeout?
No
Yes
Yes
Idle
Timeout Process

## 5. Special Considerations

NetFRAME *Wire Service Implementation Command Reference* describes the various operations executed by the Wire Service processors. However, there are additional actions that must take place for some of the control, monitoring, and feedback functions f Wire Service. These acti ns are described in this chapter.

### 5.1 Wire Service Switches

Some Wire Service operations can be controlled with toggle switches. These switches are:

- Main AC power switch
- System 5-volt power switch
- CPU reset switch
- FLASH enable switch
- NMI enable switch

These switches utilize the debounce feature in the Wire Service firmware. They are monitored periodically and an action takes place when a switch is set. In addition, when a switch is set an entry is written to the log. An entry is also written to the log when the system 5-volt power switch is cleared.

The switches and the actions taken are described in greater detail in the following sections.

#### 5.1.1 Main AC Power Switch

There are two main AC power switches, which are located in the center of the panel. They either enable or disable power for the Chassis Controller, System Recorder, and NVRAM on the Back Plane. These switches provide the AC 110-volt power supply to Raptor. This power supply is then transformed into two DC power lines, supplying power for the Raptor system. These two DC power lines are:

1. The bias 5-volt line, which provides power to the Chassis Controller and System Recorder.
2. The system 5-volt line, which provides power to the whole system, except to the Chassis Controller and System Recorder.

**Note**: The Remote Interface Board has its own power supply, which is an external 5-volt adapter.

#### 5.1.2 System 5-volt Power Switch

The system 5-volt power switch provides power to the whole system, except for the Wire Service Chassis Controller and System Recorder. You can either enable this switch locally or execute the System Master Power command. This switch is located in the upper-right corner of the panel. The main AC power switch must be enabled prior to enabling this switch.

#### 5.1.3 CPU Reset Switch

The CPU reset switch resets all main CPU functions. You can either enable this switch locally or execute the System Halt/Run command. This switch is the right-most switch under the LED Display on the panel.

#### 5.1.4 FLASH Enable Switch

The Flash enable capability to BIOS can be enabled or disabled by either toggling the Flash enable switch or executing the Flash Enable command. This switch is the middle switch under the LED Display on the panel.

### 5.1.5 NMI Enable Switch

The NMI enable switch can be enabled t take an action according to conditions in the NMI mask. You can also execute the NMI Request command to enable or disable NMI. This switch is the left-most switch under the LED Display on the panel.

## 5.2 Power Processes

There are processes for power up and power off, once the main AC power and the system 5-volt power are enabled. These processes are described in the sections below.

### 5.2.1 Initial Power Up Process

There is a start-up process once the main AC power and the system 5-volt power are enabled. The serial number of the System Board, Back Plane, canisters, and power supplies are scanned, using the Dallas Inc. one-wire serial protocol. The System Recorder processor writes the serial number data to NVRAM. The serial numbers of the Back Plane and Remote Interface are also scanned and written to NVRAM. If a component is not present, such as the Remote Interface, the length of the serial number string address for that component is zero. Refer to Section 3.4 Storing Serial Numbers on page 15, for more information on serial number data.

### 5.2.2 Power Up Process

Once the power is on, a Wire Service internal message entry is written to the log. When the power up process is enabled, Wire Service monitors the status of several system functions. These include temperature, fan speeds, and changes in the presence of canisters and power supplies. These functions are described in Section 5.3 Monitoring the System on page 29.

Both the AC and DC power supplies are monitored periodically, and any changes to the status results in an entry being written to the log. An event is sent and the system is powered off. Refer to Section 5.4 Event Queues on page 30 for more information on events. The following voltage supplies are monitored and an entry for each supply is written to the log:

- + 12 volt
- + 5 volt
- + 3.3 volt
- - 12 volt

The temperature of all the sensors on the temperature bus are monitored in the same way as the power supplies. If any of the sensors are running too hot, the system is powered off.

Non-specific, or general faults on most components are recorded in a summary bit. However, some components are specifically monitored for faults. They are:

- Fans
- Canisters
- CPU temperature

The power to Wire Service is hot-swappable. Once the system is running and you determine that a component must be unplugged, you must follow the power off process before unplugging the component.

### 5.2.3 Power Off Process

The system is powered off by disabling the main AC power. This is executed by either toggling the system master power switch or executing the System Master Power c mmand. An entry is written to the log.

## 5.3 M nit ring the System

Several system functions are periodically monitored to determine if there is a change in the status of a specific function. These are described in the following sections.

### 5.3.1 Monitoring the System Temperature

The temperature of all the sensors on the temperature bus are monitored every second. They are controlled by reading Dallas Inc. temperature transducers connected to the serial bus on the Back Plane. These sensors are read in address order. The formula is: $-25\ °C \le x \le 70°C$. There are two temperature limits:

- Shutdown limit, which is initialized to 70°C
- Warning limit, which is initialized to 55°C

Each sensor is compared to the shutdown limit. If any temperature exceeds this limit, the system is powered off. If it is lower than the shutdown limit, each sensor is then compared to the warning limit. If any temperature exceeds this limit:

- An over-limit fault is created
- A temperature LED is set
- A temperature event is sent to the system and Remote Interface
- —An entry is written to the log

If the sensors do not exceed the shutdown or warning limit, but the over-limit bit indicates that a fault has been created, a temperature event is sent to the system and Remote Interface.

### 5.3.2 Monitoring the Presence of Canisters

Several times per second, the Back Plane monitors the status of the presence of canisters. Each line is a Dallas Inc. one-wire serial bus signal connected to a Dallas Inc. serial number chip. In order to detect the presence of a canister, a reset pulse is sent by the Wire Service to detect a canister's presence pulse. If there is a change in the presence of a canister, the presence bit is updated and a canister event is sent to the system and the Remote Interface. The previous and current canister data is written to the log. If a canister is removed from the system, no further action takes place. The length of the serial number string for that canister address is set to zero. However, if a canister is installed, its serial number is read by the Dallas Inc. one-wire protocol and written to NVRAM. Refer to Section 3.4 Storing Serial Numbers on page 15, for more information on serial number data.

### 5.3.3 Monitoring the Presence of Power Supplies

Several times per second, the Back Plane monitors the status of the presence of power supplies. Each line is a Dallas Inc. one-wire serial bus signal connected to a Dallas Inc. serial number chip. In order to detect the presence of a power supply, a reset pulse is sent by the Wire Service to detect a power supply presence pulse. If there is a change in the presence of a power supply, the presence bit is updated and a power supply event is sent to the system and the Remote Interface. The previous and current power supply data is written to the log. If a power supply is removed from the system, n further action takes place. The length of the serial number string for that power supply address is set to zero. However, if a power supply is installed, its serial number is read by the Dallas Inc. one-wire protocol and written to NVRAM. Refer to Section 3.4 Storing Serial Numbers on page 15, for more information on serial number data.

### 5.3.4 M nit ring Fan Speed

Once every second, all Wire Service processors that have fans are monitored in address rder. The System Board fan low-speed limit is 30 rps (1800 rpms). The canister fan low-speed limit is 20 rps (1200 rpms).

Each fan is compared to the low-speed limit. If any fan speed falls below this limit:

- A fan fault is created
- An entry is written to the log
- The fan fault LED is set
- The speed for both fans is set to high

To reset the fan fault LED and fan speed, either reset the fields locally or through the Remote Interface, or power cycle the canister or System Board. This LED can also be reset if a fan that fails is replaced.

## 5.4 Event Queues

Events are always sent to the System Interface and Remote Interface queues. However, if the system sends an event and one or both of these interfaces are not present at power up, the system does not receive a response. When this happens, the system does not make any attempt to resend the event.

There is a command that retrieves a critical event from the system log. Usually when you retrieve a message from the log, you must read all of the log entries. However, the Critical Event Notification command triggers an alarm in the system software to read the log for this specific event and take an action. This command is described in NetFRAME *Wire Service Implementation Command Reference*.

## 5.5 LED Considerations

When the I$^2$C bus hangs up, the system fault LED blinks. Currently, it does not get reset by the hardware. Power cycle the power supply to get out of this situation.

The settings of the system LEDs are listed below:

| Fault | Settings |
|---|---|
| System | Off is OK<br>Amber is fault |
| CPU | Green is OK<br>Amber is fault |
| Temperature | Amber is fault<br>Off is OK |
| Fan | Amber is fault<br>Off is OK |
| Flash enable | Green is enable<br>Off is disable |

If there is a canister fan fault, the system fan fault is amber. However if there is a system fan fault, the system fault LED is not effected. In other words, it is not turned on. If there is a CPU thermal fault, the system fault LED is amber.

The system fault summary LED is turned on for the f ll wing:

- Canister fan fault
- System Temperature
- CPU Temperature
- CPU Internal error
- CPU Power not OK

## 5.6 Timestamp Considerations

The real-time clock (RTC) is located in the System Recorder on the Back Plane. Once it is initialized, it clicks at one-second intervals. Since it is a four-byte field, that is 32 bits, it has the capacity of recording the time for many years without having to be reset. It has battery backup power, so if the power goes off, it continues to click.

The RTC records absolute time. It does not record time in human terms. In other words, it does not reset when the time here is reset forward or back one hour. The operating system must get a reference point for its time by reading the RTC and then synchronizing it.

## 5.7 DIMM Types

This section contains tables that list the type of DIMMs, presence detect, and configuration.

*Table 2. DIMM Types*

| PD BITS | MODULE CONFIGURATION | DRAM | RE | CE | REFRESH PERIOD (ms) | |
|---|---|---|---|---|---|---|
| 4 3 2 1 | (PARITY, ECC) | ORGANIZATION | ADDR. | ADDR. | NORMAL | SLOW |
| 1 1 1 1 | No Module | | | | | |
| 0 0 0 0 | 256K X 64/72, 72 | 256K X 16/18 | 9 | 9 | 8 | 64 |
| 0 0 0 1 | 512K X 64/72, 72 | 256K X 16/18 | 9 | 9 | 8 | 64 |
| 0 0 1 0 | 512K X 64/72, 72/80 | 512K X 8/9 | 10 | 9 | 16 | 128 |
| 0 0 1 1 | 1M X 64/72, 72/80 | 512K X8/9 | 10 | 9 | 16 | 128 |
| 0 1 0 0 | 1M X 64/72, 72/80 | 1M X 1/4/16/18 | 10 | 10 | 16 | 128 |
| 0 1 0 1 | 2M X 64/72, 72/80 | 1M X 1/4/16/18 | 10 | 10 | 16 | 128 |
| 0 1 1 0 | 1M X 64/72, 72 | 1M X 16/18 | 12 | 8 | 64 | 256 |
| 1 0 0 0 | 2M X 64/72, 72 | 1M X 16/18 | 12 | 8 | 64 | 256 |
| 1 0 0 1 | 2M X 64/72, 72/80 | 2M X 8/9 | 11 | 10 | 32 | 256 |
| 1 0 1 0 | 4M X 64/72, 72/80 | 2M X 8/9 | 11 | 10 | 32 | 256 |
| 1 0 1 1 | 4M X 72 | 4M X 1/4/18 | 12** | 11** | 64 | 256 |
| 1 0 1 1 | 4M X 64, 72/80 | 4M X 4/16 | 12 | 10 | 64 | 256 |
| 1 1 0 0 | 8M X 64/72, 72 | 4M X 16/18 | 12 | 10 | 64 | 256 |
| 1 1 0 1 | 8M X 64/72, 72/80 | 8M X 8/9 | 12 | 11 | 64 | 256 |
| 1 1 1 0 | 16M X 64/72, 72/80 | 8M X 8/9 | 12 | 11 | 64 | 256 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 1 1 1 | 16M X 64/72, 72/80 | 16M X 4 | 13 | 11 | 128 | 512 |
| 0 0 0 0<br>0 0 0 1 | 16M X/72, 72<br>32M X 72, 72 | 16M X 16/18<br>16M X 16/18 | TBD*<br>TBD* | TBD*<br>TBD* | TBD*<br>TBD* | TBD*<br>TBD* |
| 0 0 1 0<br>0 0 1 1 | 32M X 64/72, 72/80<br>16M X 64/72, 72/80 | 32M X 8/9<br>32M X 8/9 | TBD*<br>TBD* | TBD*<br>TBD* | TBD*<br>TBD* | TBD*<br>TBD* |
| 0 1 0 0<br>0 1 1 1 | 64M X 64, 72/80<br>Expansion | 64M X 4 | TBD* | TBD* | TBD* | TBD*<br>TBD* |

**NOTES:**

1. * These modules using 256M devices are for reference only and will be further defined in the future.
2. 1 = NC or driven to VOH

   0 = VSS or driven to VOL
3. ** This addressing includes a redundant address to allow mixing of 12/10 (X4) and 11/11(X1) DRAMs.
4. PD and ID terminals must each be pulled up through a resistor to VDD at the next higher level assembly. PDs will either be open (NC) or driven to VSS through on-board buffer circuits.
5. IDs will either be open (NC) or connected directly to VSS without a buffer.

*Table 3. PD Speed*

| | PD7 | PD6 |
|---|---|---|
| SPEED (tRAC) | 82 | 165 |
| 80 ns | 0 | 1 |
| 70 ns | 1 | 0 |
| 60 ns | 1 | 1 |
| 50 ns | 0 | 0 |
| 40 ns | 0 | 1 |

*Table 4. Refresh Mode*

| | ID1 |
|---|---|
| REFRESH MODE | 167 |
| NORMAL | 0 |
| SELF-REFRESH | 1 |

*Table 5 Data Configuration*

| | PD8 | PD6 |
|---|---|---|
| CONFIGURATION | 166 | 83 |
| X64 | 1 | 0 |
| X72 PARITY | 1 | 1 |
| X72 ECC | 0 | 0 |
| X82 ECC | 0 | 1 |

*Table 6. EDO Detection*

| DATA ACCESS | PD5 |
|---|---|
| MODE | 81 |
| FAST PAGE | 0 |
| FP W/EDO | 1 |

## 5.8 Known Limitati ns

### 5.8.1 Hardware

Wire Service micro controllers are $I^2C$ compatible. However, they are not $I^2C$ compliant. The hardware drivers are different. They cannot have a chip that is on the $I^2C$ bus. The work-around is to externally buffer the driver.

### 5.8.2 Memory Map of Debug - NVRAM Limitations

| | | |
|---|---|---|
| • System Log | Log type | 00008000h |
| • System Interface | Queue type | 00000400h |
| • Remote Interface | Queue type | 00000400h |
| • LCD Display | Screen type | 00003C00h |

## 5.9 Password Considerations

The default password of "NETFRAME" is assigned as part of the manufacturing process for each system. This password can be overwritten locally with the Wire Service Access Password command. To change the password remotely, the remote application software must use the currently assigned password to get into the system. Once in the system, the remote application software can change it.

## 5.10 System Fan Considerations

The high-speed and low-speed of the fans are automatically controlled by Wire Service. If any of the fans fail, Wire Service detects it, and sets all fans to the high-speed with the System Board Fans command. Wire Service also sets the fan fault LED. Until the fan that failed is replaced, Wire Service takes time (a few ms) to turn all the fans to the low-speed by clearing the field with the System Board Fans command. Wire Service also turns the fan fault LED off.

## 5.11 Canister Fan Considerations

The high-speed and low-speed of the fans are automatically controlled by Wire Service. If any of the fans fail, Wire Service detects it, and sets all fans to the high-speed with the System Board Fans command. Wire Service also sets the fan fault LED. Until the fan that failed is replaced, Wire Service takes time (a few ms) to turn all the fans to the low-speed by clearing the field with the Canister Fan Speed Data command. Wire Service also turns the fan fault LED off.

## 6. Remote Interface Serial Pr toc l

The Wire Service Remote Interface serial pr tocol communicates Wire Service messages across a point-to-point serial link. This link is between a Wire Service Remote Interface processor attached to a Wire Service Remote management processor and a remote client. It encapsulates Wire Service messages in a transmission packet to provide error-free communication and link security.

The Remote Interface serial protocol uses the concept of byte stuffing. This means that certain byte values in the data stream always have a particular meaning. If that byte value is transmitted by the underlying application as data, it must be transmitted as a two-byte sequence.

The bytes that have a special meaning in this protocol are:

| | |
|---|---|
| SOM | Start of a message |
| EOM | End of a message |
| SUB | The next byte in the data stream must be substituted before processing. |
| INT | Event Interrupt |
| Data | An entire Wire Service Message |

As stated above, if any of these byte values occur as data in a message, a two-byte sequence must be substituted for that byte. The sequence is a byte with the value of SUB, followed by a type with the value of the original byte, which is incremented by one. For example, if a SUB byte occurs in a message, it is transmitted as a SUB followed by a byte that has a value of SUB+1.

The Remote Interface serial protocol uses two types of messages:

1. Requests, which are sent by remote management systems (PCs) to the Remote Interface.
2. Responses, which are returned to the requester by the Remote Interface.

The formats of these messages are:

**Request:**

| | | | | | | |
|---|---|---|---|---|---|---|
| SOM | Seq. # | TYPE | Data | ... | Check | EOM |

**Response:**

| | | | | | | |
|---|---|---|---|---|---|---|
| SOM | Seq. # | STATUS | Data | ... | Check | EOM |

Event Interrupt

| | |
|---|---|
| | INT |

Where:

| | |
|---|---|
| SOM | A special data byte value marking the start of a message. |
| EOM | A special data byte value marking the end of a message. |
| Seq. # | A one-byte sequence number, which is incremented on each request. It is stored in the response. |
| TYPE | One of the following types of requests:<br>IDENTIFY — Requests the remote interface to send back identification information about the system to which it is connected. It also resets the next expected sequence number. Security authorization does not need to be established before the request is issued.<br>SECURE — Establishes secure authorization on the serial link by checking password security data provided in the message with the Wire System password.<br>UNSECURE — Clears security authorization on the link and attempt to disconnect it. This requires security authorization to have been previously established.<br>MESSAGE — Passes the data portions of the message to the Wire Service for execution. The response from Wire Service is sent back in the data portion of the response. This requires security authorization to have been previously established.<br>POLL — Queries the status of the remote interface. This request is generally used to determine if an event is pending in the remote interface. |
| STATUS | One of the following response status values:<br>OK — Everything relating to communication with the remote interface is successful.<br>OK_EVENT — Everything relating to communication with the remote interface is successful. In addition, there is one or more events pending in the remote interface.<br>SEQUENCE — The sequence number of the request is neither the current sequence number or retransmission request, nor the next expected sequence number or new request. Sequence numbers may be reset by an IDENTIFY request.<br>CHECK — The check byte in the request message is received incorrectly.<br>FORMAT — Something about the format of the message is incorrect. Most likely, the type field contains an invalid value.<br>SECURE — The message requires that security authorization be in effect. Or, if the message has a TYPE value of SECURE, the security check failed. |
| Check | Indicates a message integrity check byte. Currently the value is 256 minus the previous bytes in the message. For example, adding all bytes in the message up, to and including the check byte should produce a result of zero (0). |
| INT | A special one-byte message sent by the Remote Interface when it detects the transition from no events pending to one or more events pending. This message can be used to trigger reading events from the remote interface. Events should be read |

| | |
|---|---|
| | until the return status changes form OK_EVENT to OK. |

## 7. Callout Script Syntax

The Wire Service callout script controls an action taken by the Remote Interface when it is requested to make a callout. The script is a compact representation of a simple scripting language that c ntrols the interaction between a modem and a remote system. Because the script keyword fields are bytes, it requires a simple compiler to translate from text to the script. A script is stored in the system recorder and is retrieved by the Remote Interface, when needed.

| Field Name | Data | Function |
|---|---|---|
| LABEL | Label Value | Establishes a label in the script. |
| GOTO | Label Value | Transfers control to a label. |
| SPEED | Speed Value | Sets the remote interface speed to the specified value. |
| SEND | Data String | Sends the data string to the serial interface. |
| TEST | Condition,<br>Label | Tests the specified condition and transfer to label if the test is true. |
| TRAP | Event,<br>Label | Establishes or removes a trap handler address for a given event. |
| SEARCH | Data String,<br>Label Value | Searches for a specific data string of the receiving buffer. If the data string is found, remove the data up to and including this string, from the buffer. Then, transfer to the label. |
| CONTROL | Control | Takes the specified control action. |
| WAIT | .1 through<br>25.5 Seconds | Delays execution of the script for the specified time. |
| EXIT | OK, FAIL | Terminates script processing and exit with a status and log result. |

**Note:** The script keyword fields are not implemented at this time.

## 8. Include File Information

The details necessary to implement this product, such as exact values for events r message types, is not c ntained in this document. These details are volatile and if included, would result in the creation f multiple c pies in different formats, by the various engineers wh implement Wire Service. In addition, it would be difficult to synchronize the code when changes are required.

There is one file, called CS9000WS.SDL. It contains all of the current values for any Wire Service related value. This file is in System Definition Language (SDL), which converts it to an H file and an INC file. The H and INC files are then used by assembly language, C, and C++ as include files, which are suitable for use in this implementation. The include file for assembly language is CS9000WS.INC, and the H file for C or C++ is CS9000WS.H. The original and current versions of this file are available on MILO\ENG\NF\GLOBAL\INCLUDE.

# 9. Event ID Codes

The Wire Service event ID codes are listed in the following sections.

## 9.1 BIOS Event ID Codes

| ID | Description |
|---|---|
| 0002 | Verify Real Mode |
| 0004 | Get CPU type |
| 0006 | Initialize System Hardware |
| 0018 | Initialize timer |
| 0024 | Set ES segment register to 4GB |
| 0008 | Initialize chipset registers with initial POST values |
| 0011 | Load alternate registers with initial POST values |
| 000C | Initialize caches to initial POST values |
| 000E | Initialize I/O |
| 0016 | BIOS ROM check |
| 0017 | Initialize external cache Before memory autosize |
| 0028 | Autosize DRAM |
| 003A | Autosize cache |
| 002A | Clear 512K base RAM |
| 002C | Test 512K base address lines |
| 002E | Test first 512K of RAM |
| 002F | Initialize external cache Before memory shadowing |
| 0038 | Shadow the system BIOS |
| 0020 | Test DRAM refresh |
| 0009 | Set in POST flag |
| 000A | Initialize CPU registers |
| 000B | Enable CPU cache |
| 0010 | Initialize power management |
| 0014 | Initialize keyboard controller |
| 001A | DMA controller initialization |
| 001C | Reset programmable interrupt controller |
| 0022 | Test keyboard controller |
| 0032 | Computer CPU speed |
| 0034 | Test CMOS RAM |
| 003C | Configure advanced chipset registers |
| 003D | Load alternate registers with CMOS values |
| 0042 | Initialize interrupt vectors 0 through 77h to the BIOS general interrupt handler |
| 0046 | verify the ROM copyright notice |
| 0047 | Initialize PCI Option ROM manager |
| 0049 | Initialize PCI bus and devices |
| 0048 | Check video configuration against CMOS |
| 004A | Initialize all video adapters in system |
| 004C | Shadow video BIOS ROM |
| 0024 | Set ES segment register to 4GB |
| 0052 | Test keyboard |
| 0054 | Initialize keystroke clicker if enabled in Setup |
| 0076 | Check for keyboard error |
| 0058 | Test for unexpected interrupts |
| 004B | Display QuietBoot screen |
| 0046 | verify the ROM copyright notice |
| 0050 | Display CPU type and speed |

| | |
|---|---|
| 005A | Display prompt "Press F2 to enter SETUP" |
| 005B | Disable CPU cache |
| 005C | Test RAM between 512K and 640K |
| 0060 | Test extended memory |
| 0062 | Test extended memory address lines |
| 0064 | Jump to UserPatch 1 |
| 0066 | Configure advanced cache registers |
| 0068 | Enable external and CPU cache |
| 006A | Display external cache size on the screen |
| 006C | Display shadow message |
| 006E | Display non-disposable segments |
| 0070 | Display error messages |
| 0072 | Check for configuration errors |
| 0074 | Test real-time clock |
| 007C | Set up hardware interrupt vector |
| 007E | The coprocessor initialization test |
| 0096 | Cleat huge ES segment register |
| 0094 | Disable A20 address line |
| 0080 | Disable on board I/O ports |
| 0085 | Display any ESCD read error and initialize PnP ISA devices |
| 0082 | Test and identify RS 232 ports |
| 0084 | Test and identify parallel ports |
| 0086 | Re-initialize on board I/O port |
| 0088 | Initialize BIO data area |
| 008C | Initialize floppy controller |
| 0090 | Initialize hard-disk controller |
| 008A | Initialize extended BIO data area |
| 008B | Setup interrupt vector and present bit in equipment byte |
| 0095 | Check and initial CD-ROM driver |
| 0093 | Build MPTABLE for multi-processor boards |
| 0092 | Jump to UserPatch 2 |
| 0098 | Search for BIOS ROM extensions |
| 009C | Setup power management |
| 009E | Enable hardware interrupt |
| 00A0 | Verify that the system clock is interrupting |
| 00A2 | Setup Numlock indicator |
| 00A4 | Initialize typematic rate |
| 00A8 | Erase F2 key prompt |
| 00AA | Scan for F2 key stroke |
| 00AC | Enter SETUP if F2 was pressed |
| 00AE | Clear in-POST flag |
| 00B0 | Check for error |
| 00B2 | POST done - prepare to boot operating system |
| 00B4 | One quick beep |
| 00B5 | Clear QuietBoot screen |
| 00B6 | Check password (Optional) |
| 00BC | Clear parity-error latch |
| 00BA | Initialize the DMI header and sub-structures |
| 00BE | If BCP option is enabled, clear the screen before booting |
| 00BF | Check virus and backup reminders |
| 008F | Get total ATA drivers in the system |
| 0091 | Initialize local-bus hard-disk controller |
| 009F | Check the total fdisks (ATA and SCSI) in the system |
| 00C0 | Try boot with INT 19 |

| | |
|---|---|
| 000F | Stub Routine |
| 0012 | Restore the contents of the CPU control word |
| 0013 | Reset the PCI bus master |
| 0036 | Vector to proper shutdown routine |
| 004E | Display the c pyright message |
| 0076 | Check for keyboard errors |
| 0094 | Disable the A20 address line |
| 0096 | Reset the segment registers to 64KB limit |
| 00BB | Stub Routine |
| 00BD | Stub Routine |
| 00D0 | Interrupt Handler Error |
| 00D2 | Unknown Error |
| 00D4 | Pending Interrupt Error |
| 00D6 | Initialize Option ROM Error |
| 00D8 | Shutdown Error |

## 9.2 Diagnostic Event ID Codes

| ID | Description | |
|---|---|---|
| 1001 | DIMM CONFIGURATION ERROR | |
| 1002 | LOCK TEST FAILED | CPU 0 |
| 1003 | LOCK TEST FAILED | CPU 1 |
| 1004 | LOCK TEST FAILED | CPU 2 |
| 1005 | LOCK TEST FAILED | CPU 3 |
| 1006 | UNCORRECTABLE MEMORY ERROR | |
| 1007 | CORRECTABLE MEMORY ERROR | |
| 1008 | SLAVE FAILED CPU 0 | |
| 1009 | SLAVE FAILED CPU 1 | |
| 100A | SLAVE FAILED CPU 2 | |
| 100B | SLAVE FAILED CPU 3 | |
| 100C | AP# PARITY ERROR | |
| 100D | RP# PARITY ERROR | |
| 100E | CORRECTABLE ERROR ON P6 BUS | |
| 100F | UNCORRECTABLE ERROR ON P6 BUS | |
| 1010 | P6 PROTOCOL VIOLATION | |
| 1011 | PCI PARITY ERROR | |
| 1012 | AERR ERROR | |
| 1013 | BERR ERROR | |
| 1014 | | |
| 1015 | | |
| 1016 | | |
| 1017 | | |
| 1018 | | |
| 1019 | MEMORY DATA ERROR | |
| 101A | MIX GEN TEST ERROR | |
| 101B | PCI WRT ERROR | |
| 101C | TIMER/APIC FAILED | |
| 101D | | |
| 101E | TIMER STOPPED | |
| 101F | | |
| 1020 | INTRAPULSE LCD WRITE ERROR | |
| 1021 | INTRAPULSE WRITE ERROR | |
| 1022 | INTRAPULSE READ ERROR | |
| 1023 | NO CANISTER | |

| | | |
|---|---|---|
| 1024 | DIAGNOSTIC START | |
| 1025 | DIAGNOSTIC STOP | |
| 1026 | CPU NMI ERROR | CPU 0 |
| 1027 | CPU NMI ERROR | CPU 1 |
| 1028 | CPU NMI ERROR | CPU 2 |
| 1029 | CPU NMI ERROR | CPU 3 |
| 102A | CPU CACHE ERROR | CPU 0 |
| 102B | CPU CACHE ERROR | CPU 1 |
| 102C | CPU CACHE ERROR | CPU 2 |
| 102D | CPU CACHE ERROR | CPU 3 |
| 102E | SLAVE COMPARE ERROR | CPU 0 |
| 102F | SLAVE COMPARE ERROR | CPU 1 |
| 1030 | SLAVE COMPARE ERROR | CPU 2 |
| 1031 | SLAVE COMPARE ERROR | CPU 3 |
| 1032 | MIX TEST COMPARE ERROR | CPU 0 |
| 1033 | MIX TEST COMPARE ERROR | CPU 1 |
| 1034 | MIX TEST COMPARE ERROR | CPU 2 |
| 1035 | MIX TEST COMPARE ERROR | CPU 3 |

| | |
|---|---|
| 1041 | INTRAPULSE EVENT CANISTER CHANGE |
| 1042 | INTRAPULSE EVENT POWER SUPPLY |
| 1043 | INTRAPULSE EVENT QUEUE |
| 1044 | INTRAPULSE EVENT TEMPERATURE |
| 1045 | INTRAPULSE EVENT AC POWER |
| 1046 | INTRAPULSE EVENT DC POWER |
| 1047 | INTRAPULSE EVENT FAN |
| 1048 | INTRAPULSE EVENT LOCK |
| 1049 | INTRAPULSE EVENT SCREEN |

### 9.3 PICs Event ID Codes

| ID | Description |
|---|---|
| 3001 | Fan #1 System Board Failure |
| 3002 | Fan #2 System Board Failure |
| 3003 | Fan #3 System Board Failure |
| 3004 | Fan #4 System Board Failure |
| 3005 | Fan #5 System Board Failure |
| 3006 | Fan #6 System Board Failure |
| 3007 | Reset System ... |
| 4001 | System CPU #1 Fault |
| 4002 | System CPU #2 Fault |
| 4003 | System CPU #3 Fault |
| 4004 | System CPU #4 Fault |
| 4005 | Flash System BIOS ... |
| 4006 | NMI Pressed ... |
| 4007 | OS timeout event |
| 6001 | Install Power Supply #1 |
| 6002 | Disconnect Power Supply #1 |
| 6003 | Install Power Supply #2 |
| 6004 | Disconnect Power Supply #2 |
| 6005 | Install Power Supply #3 |
| 6006 | Disconnect Power Supply #3 |
| 6007 | ACOK of P wer Supply #1 Changed |
| 6008 | ACOK of Power Supply #2 Changed |

| | |
|---|---|
| 6009 | DCOK f Power Supply #1 Changed |
| 600A | DCOK of Power Supply #2 Changed |
| 600B | DCOK of P wer Supply #3 Changed |
| 600C | Install Canister A |
| 600D | Disconnect Canister A |
| 600E | Install Canister B |
| 600F | Disconnect Canister B |
| 6010 | Install Canister C |
| 6011 | Disconnect Canister C |
| 6012 | Install Canister D |
| 6013 | Disconnect Canister D |
| 6014 | Temperature sensor #1 exceeds warning threshold |
| 6015 | Temperature sensor #2 exceeds warning threshold |
| 6016 | Temperature sensor #3 exceeds warning threshold |
| 6017 | Temperature sensor #4 exceeds warning threshold |
| 6018 | Temperature sensor #5 exceeds warning threshold |
| 6019 | Temperature sensor #1 exceeds shutdown threshold |
| 601A | Temperature sensor #2 exceeds shutdown threshold |
| 601B | Temperature sensor #3 exceeds shutdown threshold |
| 601C | Temperature sensor #4 exceeds shutdown threshold |
| 601D | Temperature sensor #5 exceeds shutdown threshold |
| 601E | Temperature sensor #1 NO longer in warning |
| 601F | Temperature sensor #2 NO longer in warning |
| 6020 | Temperature sensor #3 NO longer in warning |
| 6021 | Temperature sensor #4 NO longer in warning |
| 6022 | Temperature sensor #5 NO longer in warning |
| 6023 | System Power ON ... |
| 6024 | System Power OFF ... |
| 7001 | Fan #1 of Canister A Failure |
| 7002 | Fan #2 of Canister A Failure |
| 7003 | Turn ON PCI Slot #1 |
| 7004 | Turn OFF PCI Slot #1 |
| 7005 | Turn ON PCI Slot #2 |
| 7006 | Turn OFF PCI Slot #2 |
| 7007 | Turn ON PCI Slot #3 |
| 7008 | Turn OFF PCI Slot #3 |
| 7009 | Turn ON PCI Slot #4 |
| 700A | Turn OFF PCI Slot #4 |
| 8001 | Fan #1 of Canister B Failure |

| | |
|---|---|
| 8002 | Fan #2 of Canister B Failure |
| 8003 | Turn ON PCI Slot #5 |
| 8004 | Turn OFF PCI Slot #5 |
| 8005 | Turn ON PCI Slot #6 |
| 8006 | Turn OFF PCI Slot #6 |
| 8007 | Turn ON PCI Slot #7 |
| 8008 | Turn OFF PCI Slot #7 |
| 8009 | Turn ON PCI Slot #8 |
| 800A | Turn OFF PCI Slot #8 |
| 9001 | Fan #1 of Canister C Failure |
| 9002 | Fan #2 of Canister C Failure |
| A001 | Fan #1 of Canister D Failure |
| A002 | Fan #2 of Canister D Failure |

## 10. Glossary

**Back Plane** A component of the Raptor system that contains the System Recorder, Chassis Controller, and Canister Controller A through D, if present.

**BCD format** The binary-coded decimal notation used to store the serial numbers of Wire Service components.

**BIOS** Basic Input Output System, which is a software product that accesses the Wire Service through the System Bus Interface.

**callout** A script that controls an action taken by the Remote Interface. It controls the interaction between a modem and a remote system.

**Canister Controller** Wire Service processor that maintains fan speeds and faults.

**CD** Remote port carrier detect modem.

**Chassis Controller** Wire Service processor that maintains the various power and voltage supplies. It also maintains the temperature settings.

**Check Sum** A direction control byte used to ensure the integrity of a message on the wire.

**Command ID** A field that specifies the address of the processor named in the command

**CPU A Controller** Wire Service processor that maintains the speeds for the fans on the System Board. It also maintains the LCD Display.

**CPU B Controller** Wire Service processor that maintains the fault summaries and FRU status.

**CSR** Command and status register used by the System Bus Interface

**CTS** Remote port clear-to-send modem.

**Dallas Inc.** Company name of the chip used to read serial numbers of the Raptor components to NVRAM. It is a one-wire protocol.

**diagnostic program** Wire Service program that recognizes, locates and explains faults in the hardware, firmware, and software.

**DIMM** Dual interface memory module.

**DSR** Remote port data set ready modem.

**DTR** Remote port data transfer ready modem.

**EOM** End of a message field used by the Remote Interface serial protocol.

**event** An occurrence that is significant to Wire Service.

**FIFO** A queuing technique used by the System Bus Interface, in which the next item to be retrieved is the one that has been in the queue the longest.

**FLASH** An action of programming BIOS code into a memory device.

**fly-by-wire** A system in which the monitor and control connections are made by the various processor in that system, also used by Wire Service.

**FRU** Field replaceable unit.

**$I^2C$ bus** A standardserial bus that interconnects the PIC processors n the Back Plane.

**INT** Event interrupt used by the Remote Interface serial protocol.

**Inverted Slave Addr** A byte calculated in the Wire Service firmware.

**ISA bus** A bus used to transfer messages back and forth between the CPUs on the System Board.

**LCD display** The system screen maintained by the System Recorder processor.

**LED** System LEDs that display the presence of the following faults: system, CPU, temperature, fan and FLASH enable.

**log** A collection of messages stored in NVRAM.

**LSB** Low order byte of the Command ID.

**LSBit** Low order bit of the Slave Addr.

**master assert** First command request.

**MDR** Message data register, used by the System Bus Interface.

**micro controller** A processor in the Wire Service subsystem.

**MSB** High order byte of the Command ID

**MSBit** High order bit of Type.

**NMI** Non-maskable interrupt.

**NVRAM** Non volatile random access memory, located on the Back Plane.

**packet** A wrapper around a message that is transferred to the ISA bus.

**payload** Data included in a Wire Service request message.

**PIC** The name of the chip manufactured by Microchip Inc.

**PCI Slot** A slot in the cystem for PCI compliant cards.

**queue** In Wire Service, the two queues used to transfer data between Wire Service and Raptor software.

**REP IN/OUT** An X86 instruction used to load and unload data in the System Bus Interface queues.

**RTC** Real-time clock timestamp.

**RTS** Remote port request to send modem.

**Remote Interface** Wire Service processor that maintains all of the remote port modems. It also initializes the callout script and remote events.

**SBI** See System Bus Interface.

**Slave Addr** The processor identification code.

**SOM** Start of a message. This is part of the Remote Interface serial protocol.

**status** A field that specifies whether or not the command executes successfully.

**SUB** The next byte in the data stream to be substituted, used by the Remote Interface serial protoc l.

**System Board** A network f processors interconnected with an ISA bus.

**System Bus Interface** The interface between the Wire Service process rs and the CPUs on the System Board.

**System Interface** A component of the System Bus Interface.

**Time-out** A time interval allotted for certain operation to occur.

**Timestamp** Current system time applied to a log entry.

**System Recorder** A Wire Service processor that maintains data for booting the subsystems, NVRAM data, queues, and the LCD Display.

NetFRAME®

# Wire Service Implementation

# Raptor 8 Addendum

**Version 1.0**

**February 28, 1997**

Prepared for:
NetFRAME Wire Service Implementation Group
by:
Gary Liu
Ken Nguyen

Trademarks:

# Table of Contents

# 1. Preface

## 1.1 H w this Manual is Organized

This manual consists of the following:

| Chapter | Description |
|---|---|
| **2. Introduction** | Introduces the Raptor 8 functions that differ from Raptor 16. |
| **3. Differences** | Describes the differences between Raptor 16 and Raptor 8. |
| **4. Event ID Codes** | Lists the new PIC event codes. |

## 1.2 Audience

This manual is written for personnel who implement low-level drivers and SNMP agents. It is not written for end users or marketing people.

## 1.3 Documentation

NetFRAME *Wire Service Implementation Command Reference* and NetFRAME *Wire Service Implementation User's Guide* are prerequisite manuals to this manual.

## 1.4 Summary of Amendments

| | | |
|---|---|---|
| Version 1.0 | Final Version | February 28, 1997 |

## 2. Introduction

This manual describes the differences between Raptor 16 and Raptor 8. However, there is still just one set of firmware for both. The goal of Raptor 8 is to achieve compatibility with Raptor 16.

Two new commands have been added. The number of canisters has decreased. The number of fans on the System Board has decreased. The LEDs have been moved to the back panel.

## 3. Differences

The differences between Raptor 16 and Raptor 8 are described in the following sections.

### 3.1 Back Plane

There are three power supplies and two canisters. Neither the power supplies nor the canisters have serial numbers or presence signals. The serial numbers are the same as the serial number of the Back Plane. The presence signals are tied high, that is, true.

### 3.1.1 Power Supplies

There is one more power supply connected to the Back Plane. However, the Back Plane processor does not check the serial numbers of the AC power supplies. There is no Dallas Inc. chip for them. The Back Plane only checks the DC power OK signal. The Power Supply DCOK Status command monitors the power supplies to determine if they are OK. If the DC OK signals are high, the DC line is OK. If the DC OK signals are low, the DC line is not OK. In this case, it could be that a power supply is bad or does not exist. The serial numbers of these power supplies are the same as the serial number of the Back Plane.

### 3.1.2 Canisters

The four canister on Raptor 16 have been eliminated. There are only two logical canisters; there are no physical canisters. One canister is called Group A and the other is called Group B. These two groups are located on the Back Plane, and they are not detachable, as in Raptor 16. Each group consists of four slots and each slot can be set independently.

Each canister has its own PIC chip, located at CANADDx, where *x* indicates canister Group A or Group B. However, logically it is the same as Raptor 16. There are two fans for the canisters. Each chip drives the fan for that canister, that is they are controlled separately. However, the chips monitor the rpms of both fans at the same time. If the rpms fall below the expected level for either of the fans, the chip that drives that specific fan reports a fan fault. Both chips turn the fan speed to high. The fan speed data and length remain the same. Both PIC chips have identical 2-byte data. The serial numbers of these canisters are the same as the serial number of the Back Plane.

There is no LED indicator to indicate a canister fan failure. It is reported on the LCD display.

Raptor 8 has a command to turn each PCI slot on and off independently. It is the PCI Slot Power command. The hardware has a sequence of actions to turn the power on or off to each PCI slot. When this sequence of actions is complete, the Done signal is generated by the hardware. When the firmware receives a command to turn the power on or off to a PCI slot, it does not respond until the hardware signal is generated. This eliminates an additional delay, a delay that is currently in Raptor 16.

### 3.2 System Board

The CPU A Controller still indicates LED faults, however they are located on the back of the system. In addition, they are now shown on the LCD display. The CPU B Controller sends a message to the CPU A Controller indicating system faults.

Functions of the System Board are the same for Raptor 8. However, Raptor 8 has only four fans on the System Board. These fans are the same as the Raptor 16 canister fans. Four bytes are used to report fan status.

## 3.3 LED Modificati ns

All of the LEDs are located n the back of the system. Therefore, error conditions are displayed on the LCD display, with the System Fault Status command.

## 3.4 Commands

The following commands have been added to Raptor 8. They are described in the following sections.

### 3.4.1 System Fault Status (WS_LCD_MSG)

You can use this command to report on the status of system faults. The appearance of the LCD Display is slightly different in Raptor 8. Most of the states are monitored by the CPU B Controller, so this command is used to transfer fault status from the CPU B Controller to the CPU A Controller LCD display. This command is issued by the CPU A Controller and is a write-only command.

There is 1 data byte for this command. Bits 0 through 5 are used for various operations. They use positive logic. The bits assigned to the various faults are listed below:

| Bit | Fault |
|---|---|
| 0 | FLASH_ENABLE |
| 1 | CPU_FAULT |
| 2 | TEMP_FAULT |
| 3 | FRU_FAULT |
| 4 | CANA_FAN_FAULT |
| 5 | CANB_FAN_FAULT |

The following fault status indicators are shown on the LCD display of the CPU A Controller:

| Fault | Message |
|---|---|
| <FLS_ENA> | Flash_Enable |
| <CPU_FLT> | CPU_Fault |
| <TEM_FLT> | Temp_Fault |
| <FRU_FLT | FRU_Fault |
| <CNF_FLT> | Can_Fan_Fault |
| <CBF_FLT> | SB_Fan_Fault |

Error codes for the Status field are described in NetFRAME *Wire Service Implementation Command Reference.*

## WRITE

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 02 |
| Command ID (LSB) | 07 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 01 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FC |

### 3.4.2 PCI Slot Power (WS_PCI_SLOT)

You can use this command to query the status of the PCI slots in the two logical canisters, and turn the power on and off. One canister is called Group A (Canister A) and the other is called Group B (Canister B). Each group has four PCI slots. Refer to NetFRAME *Wire Service Implementation User's Guide* or *Command Reference* to determine the slave address (**Slave Addr**) for these groups.

This command is issued to one of the Canister Controllers. A reply returned from a write operation does not mean the operation is complete. A write operation sets a bit, however it is not set at that particular second. Since there is a time lag, you should execute a read operation to verify the results.

#### *3.4.2.1 Write Operation*

Bits 0 through 3 indicate the PCI slot number for a group, as shown below:

| Bit | PCI Slot # |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 3 |
| 3 | 4 |

Bit 7 indicates clear or set, as shown below:

| Bit 7 | State |
|---|---|
| 0 | Off |
| 1 | On |

Bits 4 through 6 are not used.

#### *3.4.2.2 Read Operation*

Bits 0 through 3 indicate the PCI slot number for a group, as shown below:

| Bit | PCI Slot # |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 3 |
| 3 | 4 |

Bits 4 through 7 are not used. If a bit is set, the power to the corresponding PCI slot is on. If a bit is clear, the power to the corresponding PCI slot is off.

Error codes for the **Status** field are described in NetFRAME *Wire Service Implementation Command Reference.*

## READ

Request:

| Field | Value |
|---|---|
| Slave Addr | 40 |
| Type | 82 |
| Command ID (LSB) | 04 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 41 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DF |

## WRITE

Request:

| Field | Value |
|---|---|
| Slave Addr | 40 |
| Type | 02 |
| Command ID (LSB) | 04 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 41 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DF |

# 4. Event ID Codes

## 4.1 PICs Event ID Codes

| ID | Description |
|---|---|
| 7003 | Turn ON PCI Slot # 1 |
| 7004 | Turn OFF PCI Slot # 1 |
| 7005 | Turn ON PCI Slot # 2 |
| 7006 | Turn OFF PCI Slot # 2 |
| 7007 | Turn ON PCI Slot # 3 |
| 7007 | Turn OFF PCI Slot # 3 |
| 7009 | Turn ON PCI Slot # 4 |
| 700A | Turn OFF PCI Slot # 4 |
| 8003 | Turn ON PCI Slot # 5 |
| 8004 | Turn OFF PCI Slot # 5 |
| 8005 | Turn ON PCI Slot # 6 |
| 8006 | Turn OFF PCI Slot # 6 |
| 8007 | Turn ON PCI Slot # 7 |
| 8007 | Turn OFF PCI Slot # 7 |
| 8009 | Turn ON PCI Slot # 8 |
| 800A | Turn OFF PCI Slot # 8 |

NetFRAME®

# Wire Service Implementation

## Command Reference

**Version 1.0**

**February 28, 1997**

Prepared for:

NetFRAME Wire Service Implementation Group

by:

Gary Liu

Ken Nguyen

Trademarks:

# Table of Contents

# 1. Preface

## 1.1 How this Manual is Organized

This manual consists of the following chapters:

| Chapter | Description |
|---|---|
| 2. Introduction | Introduces the Wire System commands. |
| 3. Command Protocol | Describes the format of the commands. |
| 4. Commands | Lists the commands in processor order, gives a brief description, and shows the specific formats for each command. |

## 1.2 Audience

This manual is written for personnel who implement low-level drivers and SNMP agents. It is not written for end users or marketing people.

## 1.3 Documentation

The NetFRAME *Wire Service Implementation User's Guide* is a companion manual. You can use that manual to learn about using Wire Service.

A limited knowledge of I$^2$C protocol is a prerequisite to understanding Wire Service Protocols. Refer to *The I$^2$C-bus and How to Use It*, (Philips Semiconductor, January 1992) for more information on I$^2$C protocol. (Ken Nguyen has a copy of this document.)

This manual implements the Wire Service architecture. Refer to *Raptor Wire Service Architecture, Version 1.3*, (NetFRAME, October 1996) for details on this architecture. It contains a section that describes the physical signal connections to the Wire Service processors. It also contains detailed diagrams of the Wire Service System, and Wire Service Interface Programming State Diagram.

## 1.4 Summary of Amendments

| | | |
|---|---|---|
| Version 1.0 | Initial Version | December 17, 1996 |
| | Final Version | February 28, 1997 |

## 2. Introduction

All of the Wire Service commands conform to the $I^2C$ command format, including addressing, and read and write specifications. These commands use 7-bit addressing.

Wire Service implements two types of transactions:

- Memory-Read
- Memory Write

These transactions consist of $I^2C$ commands. Each transaction is made up of two back-to-back $I^2C$ commands with a repeated START condition between them. Keep in mind, however, that this results in the inability to rearbitrate the bus. The first command is always a write, which is master to slave. The second command is a read, which is slave to master. Any processor can originate a command, or be a master, and any processor can respond to a command, or be a slave.

## 3. C mmand Prot c l

The command, diagnostic, monitoring, and logging functions f Wire Service are accessed through the common I$^2$C bus protocol. The processors that comprise Wire Service utilize this bus for c mmunicati n. They are interconnected by this bus.

The I$^2$C bus protocol uses an address in Wire Service memory as the means of identifying the various commands. Any function can be queried by generating a "read" request, which has its address as part of its protocol format. Conversely, a function can be executed by "writing" to an address specified in the protocol format. Any Wire Service processor can initiate read and write requests by sending a message on the I$^2$C bus to the processor responsible for that function.

Multiple simultaneous read requests can result in errors, particularly read requests to the log. Several sequence numbers may be lost in a string of messages. You should use I$^2$C bus contention to ensure that multiple read requests are not simultaneous.

This protocol includes data types as part of the address data. It implements a separate address space for each data type. Refer to NetFRAME *Wire Service Implementation User's Guide* for more information on data types. The ability to use separate address spaces allows for compact internal storage. It also allows you to create unique and more complex data types, which are better suited to specific functions.

### 3.1 Generic Format for Command Protocol

A generic protocol format is used for the Wire Service commands. The formats are shown as tables for read requests and responses, and write requests and responses. These tables are shown in Section 3.1.1 Protocol Tables, on page 2. The fields in this protocol are described in Section 3.1.2 Protocol Field Descriptions, on page 5.

Both read and write requests and responses must be initiated. This is indicated by a Master Asserts START. For example, an external processor wants to set the temperature of all sensors. This is initiated with the WS_SYS_TEMP_DATA command. The master writes this command to the slave. For this specific transaction, the slave receives command data from the master. This data is transmitted to the slave, byte by byte. That is, byte 0 is transmitted, then byte 1, and so on, until byte N+5 has been transmitted. The slave then transmits the requested data back to the master. The Master Repeats START and receives (reads) the data.

### 3.1.1 Protocol Tables

A generic read request and response table, and a generic write request and response table are shown in Table 1. Generic Format Protocol on page 4. The byte fields that are grey indicate that the contents of the byte were calculated in the Wire Service firmware.

A Wire Service read and write request consists of a payload, a message, and a packet. Payload is the data included in the request. Referring to Table 1, payload for a read request is byte 4, and for a read response, it is byte 1 through byte N+1. Payload for a write request is byte 4 through byte N+4, and for a write response it is byte 1. Message is a wrapper around this data. In addition to the data, it includes **Slave Addr**, **LSBit**, **MSBit**, **Type**, **Command ID (LSB and MSB)**, and **Status**. Packet is a wrapper around a message that is transferred to the ISA bus. It includes **Check Sum** and the **Inverted Slave Addr** fields.

*Table 1. Generic Format Protocol*

## READ

Master Asserts START (Request)

| Offset | | |
|---|---|---|
| Byte 0 | Slave Addr<br>(7 bits) | 0<br>LSBit |
| Byte 1 | MSBit (1) | Type |
| Byte 2 | Command ID (LSB) | |
| Byte 3 | Command ID (MSB) | |
| Byte 4 | Read Request Length (N) | |
| Byte 5 | Check Sum | |

Master Repeats START (Response)

| Offset | | |
|---|---|---|
| Byte 0 | Slave Addr<br>(7 bits) | 1<br>LSBit |
| Byte 1 | Read Response Length (N) | |
| Byte 2 | Data Byte 1 | |
| : | : | |
| Byte N+1 | Data Byte N | |
| Byte N+2 | Status | |
| Byte N+3 | Check Sum | |
| Byte N+4 | Inverted Slave Addr | |

## WRITE

Master Asserts START (Request)

| Offset | | |
|---|---|---|
| Byte 0 | Slave Addr<br>(7 bits) | 0<br>LSBit |
| Byte 1 | MSBit (0) | Type |
| Byte 2 | Command ID (LSB) | |
| Byte 3 | Command ID (MSB) | |
| Byte 4 | Write Request Length (N) | |
| Byte 5 | Data Byte 1 | |
| : | : | |
| Byte N+4 | Data Byte N | |
| Byte N+5 | Check Sum | |

Master Repeats START (Response)

| Offset | | |
|---|---|---|
| Byte 0 | Slave Addr<br>(7 bits) | 1<br>LSBit |
| Byte 1 | Write Response Length (0) | |
| Byte 2 | Status | |
| Byte 3 | Check Sum | |
| Byte 4 | Inverted Slave Addr | |

### 3.1.2 Protoc l Field Descripti ns

The protocol fields are described in the following table. These fields can be modified only by Wire Service firmware.

| FIELD | DESCRIPTION |
|---|---|
| **Slave Addr** | Specifies the processor identification code. This field is 7 bits wide. Bit [7...1]. |
| **LSBit** | Specifies what type of activity is taking place. If LSBit is clear (0), the master is writing to a slave. If LSBit is set, it the master is reading from a slave. |
| **MSBit** | Specifies the type of command. It is bit 7 of byte 1 of a request. If this bit is clear (0), this is a write command. If it is set (1), this is a read command. |
| **Type** | Specifies the data type of this command, such as bit or string. |
| **Command ID (LSB)** | Specifies the least significant byte of the address of the processor. |
| **Command ID (MSB)** | Specifies the most significant byte of the address of the processor. |
| **Length (N)** | |
| **Read Request** | Specifies the length of the data that the master expects to get back from a read response. The length, which is in bytes, does not include the **Status**, **Check Sum**, and **Inverted Slave Addr** fields. |
| **Read Response** | Specifies the length of the data immediately following this byte, that is byte 2 through byte N+1. The length, which is in bytes, does not include the **Status**, **Check Sum**, and **Inverted Slave Addr** fields. |
| **Write Request** | Specifies the length of the data immediately following this byte, that is byte 2 through byte N+1. The length, which is in bytes, does not include the **Status**, **Check Sum**, and **Inverted Slave Addr** fields. |
| **Write Response** | Always specified as 1. |
| **Data Byte 1**<br>**:**<br>**Data Byte N** | Specifies the data in a read request and response, and a write request. |
| **Status** | Specifies whether or not this command executes successfully. A non-zero entry indicates a failure. Status codes are described in Section 3.2 Wire Service Status Codes on page 7. |
| **Check Sum** | Specifies a direction control byte to ensure the integrity of a message on the wire. This byte is calculated in the Wire Service firmware. |
| **Inverted Slave Addr** | Specifies the **Slave Addr**, which is inverted. This byte is calculated in the Wire Service firmware. |

### 3.1.3 W rking with the Fields

The values used in the protocol are derived from the address of a specific processor. This address consists f three parts:

1. Processor identification code
2. Data type
3. Subaddress

The address is 4 bytes in length and is in hexadecimal notation, as follows:

**PIDTALAMh**

Where:

**PI** is Processor ID

**DT** is Data type

**AL** is First byte of the 2-byte subaddress, that is **Command ID**. It is the least significant byte, or LSB.

**AM** is Second byte of the 2-byte subaddress, that is **Command ID**. It is the most significant byte, or MSB.

**h** is Hexadecimal notation.

#### *3.1.3.1 Slave Addr*

The identification codes for the processors are listed in the following table. The processor identification (PI) is used to determine the **Slave Addr**.

To determine the **Slave Addr** value for a <u>request</u>, shift all the bits of the **PI** to the left one bit. For example, the processor ID for the chassis controller is 02h, so after the bits are shifted, the **Slave Addr** is 04h. To determine the **Slave Addr** value for a <u>response</u>, increment the shifted value by one (1). In this case, the response **Slave Addr** is 05h.

| Processor Name | PI |
|---|---|
| System Recorder | 01h |
| Chassis Controller | 02h |
| CPU A Controller | 03h |
| CPU B Controller | 04h |
| System Interface | 10h |
| Remote Interface | 11h |
| Canister Controller A | 20h |
| Canister Controller B | 21h |
| Canister Controller C | 22h |
| Canister Controller D | 23h |

### *3.1.3.2 Type*

Valid data types (**DT**) for the commands are listed below. They determine the value specified in the **Type** field. The value for a write request is the value specified in **DT**. The value for a read request is the value specified in **DT**, with bit 7 set.

| DATA TYPE | READ | WRITE |
|---|---|---|
| Bit | 81h | 01h |
| Byte | 82h | 02h |
| String | 83h | 03h |
| Log | 84h | 04h |
| Event | 85h | 05h |
| Queue | 86h | 06h |
| Byte Array | 87h | 07h |
| Lock | 88h | 08h |
| Screen | 89h | 09h |

### *3.1.3.3 Command ID*

The subaddress (**ALAM**) is used to determine the **Command ID**. The first byte, or **AL**, is the least significant byte, or LSB. The second byte, or **AM**, is the most significant byte, or MSB.

## 3.2 Wire Service Status Codes

If the Wire Service **Status** byte contains a non-zero number, an error has occurred. The error codes are described as follows:

| CODE | DESCRIPTION |
|---|---|
| 1 | The slave did not respond to a request initiated by a master. Resubmit the command with a correct **Slave Addr**. |
| 2 | The slave did not have the data type or address specified in a request sent by the master. Resubmit the command with the correct data **Type** or **Command ID**. |
| 3 | The message or response is not valid. The **Slave Addr** is invalid. The master does not recognize the response. No further messages are sent for this transaction. |
| 4 | The message could not be completely sent or received. The master does not recognize the response. No further messages are sent for this transaction. |
| 5 | Message data in the **Check Sum** field was received incorrectly. The request should be resent, if possible. |
| 6 | The slave operation is not valid, such as a write request to a read-only command. |
| 7 | The slave responded that there was no data at the specified address for a queue or a log. |

## 4. Commands

The commands in this manual are organized by Wire Service processor and further by address. The processors are listed below. Refer to Section 3.1 Generic Format for Command Protocol on page 2, for a detailed explanation of the general format and syntax for the commands. Keep in mind that numerical data in the examples are hexadecimal.

## 4.1 Wire Service C mm n Commands

The commands in this section are used by all the processors in Wire Service. Y u can use these commands to query the system for a description of the various processors, and their firmware revisions.

### 4.1.1 Processor Type and Description (WS_DESCRIPTION)

You can use this command to query the description of a specific processor in Wire Service. There is one command for each processor. The value, nn, for the **Slave Addr** and **Inverted Slave Addr** is the ID for a specific processor. This is a read-only command.

Error codes for the Status field Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | nn |
| Type | 83 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 40 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | nn |
| Length | 40 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| CheckSum | |
| Inverted Slave Addr | nn |

### 4.1.2 Wire Service Software Revisi n (WS_REVISION)

You can use this command t query the date and description of firmware revisions made t Wire Service. There is one command for each processor. The value for the **Slave Addr** and **Inverted Slave Addr** is the ID f r a specific processor. This is a read-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | nn |
| Type | 83 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | 20 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | nn |
| Length | 20 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | nn |

## 4.2 System Recorder Process r C mmands

The System Recorder process r maintains data for booting the subsystem, NVRAM data, queues, and the system screen. It also maintains serial numbers for the canisters, and other components that have serial numbers. The system log and timestamp data are also maintained by this processor.

### 4.2.1 AC Power Available (WS_POWERUP_HOLD)

(NOT USED)

### 4.2.2 Callout on Time-out (WS_WDOG_CALLOUT)

You can use this command to query, initialize, and modify the setting for a callout request on a watchdog time-out. If this bit is set, a callout is initiated for this time-out.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 81 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 01 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.3 Reset on Time-out (WS_WDOG_RESET)

You can use this command to query, initialize, and modify the setting for a watch dog time-out. If this bit is set, the system is reinitialized.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 81 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 01 |
| Data Byte 1 | 01 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 01 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.4 Reset NVRAM Data (WS_NVRAM_RESET)

You can use this command as a trigger t reset NVRAM data. When "0x5a" is written to this byte, all f the NVRAM data is cleared. This byte should be the last to be cleared, in order to indicate that reinitializati n is complete.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 82 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 02 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | 5A |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.5 System Boot Flag 1 (WS_BOOTFLAG1)

You can use this command to query, initialize, and modify the setting for the behavior of the system. This flag is reserved for use by BIOS.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 82 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 01 |
| Data Byte 1 | 01 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 02 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.6 System Boot Flag 2 (WS_BOOTFLAG2)

You can use this command to query, initialize, and modify the setting for the behavior f the system. It is reserved for use by the diagnostic program. The description of the bits are shown below:

| Bit # | Use | |
|---|---|---|
| 0 | Diagnostic stop error | |
| 1 | Extensive diagnostic | |
| 2 | Quick diagnostic | |
| 3, 4 | Not used | |
| 5, 6, 7 | 000 | 200 MHz |
| | 001 | 133 MHz |
| | 010 | 167 MHz |
| | 011 | 200 MHz |
| | 100 | 233 MHz |
| | 101 | 267 MHz |
| | 110 | Not used |
| | 111 | Not used |

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

### READ

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 82 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 01 |
| Data Byte 1 | 01 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### WRITE

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 02 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.7 System Boot Flag 3 (WS_BOOTFLAG3)

You can use this command to query, initialize, and modify the setting for the behavior of the system. It is used by BIOS, and diagnostic and operating systems. The valid values are defined by those systems.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 82 |
| Command ID (LSB) | 04 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 01 |
| Data Byte 1 | 01 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 02 |
| Command ID (LSB) | 04 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.8 System Boot Flag 4 (WS_BOOTFLAG4)

You can use this command t query, initialize, and modify the setting for the behavior f the system. This flag is reserved for the diagnostic program. When this byte is clear diagnostics are run on Reset. When this byte is set, BIOS is run on Reset.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 82 |
| Command ID (LSB) | 05 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 01 |
| Data Byte 1 | 01 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 02 |
| Command ID (LSB) | 05 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.9 Size of NVRAM (WS_SYS_XDATA_KBYTES)

You can use this command to query the size of external data in NVRAM. This byte maintains the size of external data in NVRAM, that is WS_SYS_XDATA, in kilobytes. This is a read-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 82 |
| Command ID (LSB) | 06 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.10 NVRAM Fault (WS_NVRAM_FAULTS)

You can use this command to query the presence of faults for NVRAM data. This is a read-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 82 |
| Command ID (LSB) | 07 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.11 NVRAM Data (WS_SYS_XDATA)

Y u can use this command to query, initialize, and modify the setting for external data stored in NVRAM. Wire Service merely maintains this byte array: it does not use it.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 87 |
| Command ID (LSB) | 00 |
| Command ID (MSB) | 00 |
| Length | FF |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | FF |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 07 |
| Command ID (LSB) | 00 |
| Command ID (MSB) | 00 |
| Length | FF |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.12 System Log (WS_SYS_LOG)

You can use this command to query, initialize, and modify the setting for the system log in NVRAM. Refer to NetFRAME *Wire Service Implementation User's Guide* for more information about l g data types.

Error codes f r the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 84 |
| Command ID (LSB) | 00 |
| Command ID (MSB) | 00 |
| Length | FF |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | FF |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 04 |
| Command ID (LSB) | 00 |
| Command ID (MSB) | 00 |
| Length | FF |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.13 Rem te Interface Queu (WS_RI_QUEUE)

You can use this command to queue data being transferred to the Remote Interface processor. Refer to NetFRAME *Wire Service Implementation User's Guide* for more information about queues.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 86 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | FF |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | FF |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 06 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | FF |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.14 System Interface Queue (WS_SI_QUEUE)

You can use this command to queue data going to the System Interface processor. Refer to NetFRAME *Wire Service Implementation User's Guide*, for more information about queues.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 86 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | FF |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | FF |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 06 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | FF |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.15 System Screen (WS_SYS_SCREEN)

You can use this command t query, initialize, and modify the copy of the most recent character-mode screen from the system video display. The maximum value for Length is F0. Refer to NetFRAME *Wire Service Implementation User's Guide* for more inf rmati n about screens.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 89 |
| Command ID (LSB) | 00 |
| Command ID (MSB) | 00 |
| Length | F0 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | F0 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 09 |
| Command ID (LSB) | 00 |
| Command ID (MSB) | 00 |
| Length | F0 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.16 Call ut Script (WS_CALLOUT_SCRIPT)

You can use this command to query and initiate a callout script. It controls an action taken by the Remote Interface processor, when it is requested to make a callout.

Error codes f r the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 83 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | FF |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | FF |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 03 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | FF |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.17 Access Password (WS_PASSWORD)

You can use this command to query, initialize, and modify the access password. It is used for remote access to Wire Service.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 83 |
| Command ID (LSB) | 04 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 03 |
| Command ID (LSB) | 04 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.18 Back Plane Serial Data (WS_SYS_BP_SERIAL)

You can use this command t query the current serial number of the Back Plane. Even though this is a read and write c mmand, the serial number should not be modified.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 83 |
| Command ID (LSB) | 05 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 03 |
| Command ID (LSB) | 05 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.19 Canister A Serial Data (WS_SYS_CAN_SERIAL1)

You can use this command t query the current serial number for Canister A. The length of this string is zero if a canister is not present. Even though this is a read and write command, the serial number should not be modified.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 83 |
| Command ID (LSB) | 06 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 03 |
| Command ID (LSB) | 06 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.20 Canister B Serial Data (WS_SYS_CAN_SERIAL2)

You can use this c mmand to query the current serial number for Canister B. The length of this string is zero if a canister is not present. Even though this is a read and write command, the serial number sh uld not be modified.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 83 |
| Command ID (LSB) | 07 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 03 |
| Command ID (LSB) | 07 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.21 Canister C Serial Data (WS_SYS_CAN_SERIAL3)

Y u can use this command to query the current serial number f r Canister C. The length f this string is zero if a canister is not present. Even though this is a read and write command, the serial number should not be modified.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 83 |
| Command ID (LSB) | 08 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 03 |
| Command ID (LSB) | 08 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.22 Canister D Serial Data (WS_SYS_CAN_SERIAL4)

Y u can use this command to query the current serial number for Canister D. The length of this string is zero if a canister is not present. Even th ugh this is a read and write command, the serial number should not be modified.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 83 |
| Command ID (LSB) | 09 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 03 |
| Command ID (LSB) | 09 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.23 Remote Interface Serial Data (WS_SYS_RI_SERIAL)

You can use this command to query the current serial number for the Remote Interface processor. Even though this is a read and write command, the serial number should not be modified.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 83 |
| Command ID (LSB) | 16 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 03 |
| Command ID (LSB) | 16 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.24 System Board Serial Data (WS_SYS_SB_SERIAL)

You can use this command to query the current serial number for the System Board. Even though this is a read and write command, the serial number should not be modified.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 83 |
| Command ID (LSB) | 17 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 03 |
| Command ID (LSB) | 17 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.25 P wer Supply 1 Serial Data (WS_SYS_PS_SERIAL1)

You can use this command to query the current serial number for the first power supply. The length f this string is zero if this power supply is not present. Even though this is a read and write command, the serial number should not be modified.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 83 |
| Command ID (LSB) | 18 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| 10 | |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 03 |
| Command ID (LSB) | 18 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.26 Power Supply 2 Serial Data (WS_SYS_PS_SERIAL2)

You can use this command t query the current serial number for the second power supply. The length of this string is zero if this power supply is not present. Even though this is a read and write command, the serial number should not be modified.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 83 |
| Command ID (LSB) | 19 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| 10 | |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 03 |
| Command ID (LSB) | 19 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.27 P wer Supply 3 Serial Data (WS_SYS_PS_SERIAL3)

(NOT USED)

### 4.2.28 System ID (WS_NAME)

You can use this command to query, initialize, and modify the name that identifies Wire Service. A user can define a unique name for this system. The default is no name.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 83 |
| Command ID (LSB) | 1B |
| Command ID (MSB) | 00 |
| Length | 20 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 20 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 03 |
| Command ID (LSB) | 1B |
| Command ID (MSB) | 00 |
| Length | 20 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.29 System Boot Devices (WS_BOOTDEVS)

(NOT USED)

### 4.2.30 System Log Timestamp (WS_SYS_LOG_CLOCK)

You can use this command to query the current time from the system log timestamp clock. This is a read-only command. Refer to NetFRAME *Wire Service Implementation User's Guide* for more information on the timestamp clock.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 83 |
| Command ID (LSB) | 1D |
| Command ID (MSB) | 00 |
| Length | 04 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 04 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.31 System Log C unt (WS_SYS_LOG_COUNT)

You can use this command to query the number of entries in the system log. This is a read-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 83 |
| Command ID (LSB) | 1E |
| Command ID (MSB) | 00 |
| Length | 02 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 02 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

### 4.2.32 Critical Event Notification (WS_EVENT_IDnn)

You can use this command to retriev  a critical event that has been written t  NVRAM. This type of event triggers an OS alarm. There are ten Critical Event Notification commands. They are all the same except for the ID number in the command and the **C  mmand ID**. In the command, 'nn' represents 01 thr  ugh 0A, and **Command ID** is 20 through 29.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 83 |
| Command ID (LSB) | 20-29 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 02 |
| Type | 03 |
| Command ID (LSB) | 20-29 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 03 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FD |

## 4.3 Chassis C ntroller C mmands

The Chassis Controller maintains the various power and voltage supplies. It also maintains the temperature settings, and system reinitialization.

### 4.3.1 System Master Power (WS_SYS_POWER)

You can use this command to query, initialize, and modify the current system master power setting of the Raptor system, except for the Chassis Controller and the System Recorder. When this bit is set, power is turned on, and the system run line is activated, that is WS_SYS_RUN is set. An entry is written to the log. When this bit is cleared, power is shut off and an entry is written to the log.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 81 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FB |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 01 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FB |

### 4.3.2 System Master Power Request (WS_SYS_REQ_POWER)

(NOT USED)

### 4.3.3 +12 V lt Main Supply (WS_BP_P12V)

You can use this command to query the analog measurement of the positive 12 volt main supply. This is a read-only command.

| Voltage | A/D Byte |
|---|---|
| 12v | 90h (144d) |

X12v = Data reads from WS_BP_P12V

Vmeasure = (12v * X12v) / (90h)

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 82 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FB |

### 4.3.4 +3.3 V lt Main Supply (WS_BP_P3V)

You can use this command to query the anal g measurement f the positive 3.3 volt main supply. This is a read-only command.

| Voltage | A/D Byte |
|---|---|
| 3.3v | AFh (175d) |

X3.3v = Data reads from WS_BP_P3V

Vmeasure = (3.3v * X3.3v) / (AFh)

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 82 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FB |

### 4.3.5 -12 V lt Main Supply (WS_BP_N12V)

You can use this command to query the analog measurement of the negative 12 volt main supply. This is a read-only command.

| V ltage | A/D Byte |
|---|---|
| -12v | 92h (146d) |

Xneg12v = Data reads from WS_BP_N12V

Vmeasure = (-12v * Xneg12v) / (92h)

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 82 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FB |

### 4.3.6 +5 V lt Main Supply (WS_BP_P5V)

You can use this c mmand to query the anal g measurement of the positive 5 volt main supply. This is a read-only command.

| V ltage | A/D Byte |
|---|---|
| 5v | A3h (163d) |

X5v = Data reads from WS_BP_P5V

Vmeasure = (5v * X5v) / (A3h)

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 82 |
| Command ID (LSB) | 04 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | FB |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FB |

### 4.3.7 A/D V ltage Referenc (WS_BP_VREF)

You can use this command t query the analog measurement of the A/D voltage reference. This is a read-only command.

| V ltage | A/D Byte |
|---|---|
| Vref (5v) | FFh (255d) |

Xvref = Data reads from WS_BP_VREF

Vmeasure = (Vref * Xvref) / (FFh)

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 82 |
| Command ID (LSB) | 05 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FB |

### 4.3.8 Back Plane System (WS_SYS_BP_TYPE)

You can use this command to query the type of Back Plane system. This byte is always clear, indicating that there are four small canisters. This is a read-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 82 |
| Command ID (LSB) | 06 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FB |

### 4.3.9 Canister Presence (WS_SYS_CAN_PRES)

You can use this command to query the presence f ne, or all of the canisters. When the system is powered up, this byte is read to determine the presence of canisters. The system is monitored periodically to determine if a new canister has been installed. When the system recognizes a new canister, its serial number is written to WS_SYS_CAN_SERIALn. An entry is written to the log, and an event is sent. This is a read-only command.

| Bit Set | Canister Presence |
|---|---|
| 0 | Canister A |
| 1 | Canister B |
| 2 | Canister C |
| 3 | Canister D |

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 82 |
| Command ID (LSB) | 07 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FB |

### 4.3.10 Power Supply ACOK Status (WS_SYS_PS_ACOK)

You can use this command t query the status of the AC power supplies to determine if they are good or bad. If the status is set to "1" the power supply is OK. These power supplies are detachable. This is a read-only command.

| Bit Set | Power Supply Presence |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 3 |

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 82 |
| Command ID (LSB) | 08 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FB |

### 4.3.11 Power Supply DCOK Status (WS_SYS_PS_DCOK)

You can use this command to query the status of the DC power supplies to determine if they are good r bad. If the status is set to "1" the power supply is OK. These power supplies are detachable. This is a read-only command.

| Bit Set | Power Supply Presence |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 3 |

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 82 |
| Command ID (LSB) | 09 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FB |

### 4.3.12 Power Supply Presence (WS_SYS_PS_PRES)

You can use this command to query the presence of one r two power supplies. When the system is powered up, this byte is read to determine the presence f one or two power supplies. The system is monitored periodically to determine if a new power supply has been installed. When the system rec gnizes a new power supply, its serial number is written to WS_SYS_PS_SERIALn. An entry is written to the log, and an event is sent. If bit 0 is set, power supply 1 is present. If bit 1 is set, power supply 2 is present. This is a read-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 82 |
| Command ID (LSB) | 0A |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FB |

### 4.3.13 Delay Reset/Run (WS_SYS_RSTIMER)

(NOT USED)

### 4.3.14 Shutdown Temperature (WS_SYS_TEMP_SHUT)

You can use this command to query, initialize, and modify the setting for the shutdown temperature on the Back Plane. The initialized default shutdown temperature is 70°C. If the temperature of any of the processors exceeds this value, the system master power is shut off, that is WS_SYS_POWER is cleared. An entry is written to the log.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 82 |
| Command ID (LSB) | 0C |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FB |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 02 |
| Command ID (LSB) | 0C |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FB |

### 4.3.15 Warning Temperature (WS_SYS_TEMP_WARN)

You can use this command to query, initialize, and modify the setting for the current warning temperature on the Back Plane. The initialized warning temperature is 55°C. If the temperature of any of the processors exceeds this value, an warning fault is created, that is WS_SYS_OVERTEMP is set. An entry is written to the log.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 82 |
| Command ID (LSB) | 0D |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FB |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 02 |
| Command ID (LSB) | 0D |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FB |

### 4.3.16 System Watchdog Timer (WS_SYS_WDOG)

(NOT IMPLEMENTED AT THIS TIME)

### 4.3.17 Temperature f all Sens rs (WS_SYS_TEMP_DATA)

You can use this command to query, initialize, and modify the temperatures of all the sensors. The Dallas Inc. temperature transducers connected to the serial bus are read in address order. This is a read-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 83 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 05 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 05 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FB |

### 4.3.18 OS Timeout Res lution (WS_OS_RESOLUTION)

You can use this command to query, initialize, and modify the OS watchdog timeout resolution. It is used in conjunction with the OS Timeout Counter command, described in the next section.

At power up, the default resolution is 125 ms. When an OS Timeout Counter command is issued with non-zero data, the OS timeout counter begins to count down based on the resolution specified in the OS Timeout Resolution command. OS should reset this counter periodically with the OS Timeout Command. If OS should miss it, this counter counts down to zero and creates an OS TIMEOUT event (0A), logs it, and transfers it through the System Bus Interface and Remote Interface. Resolution is specified in **Data Byte 1** as follows:

| Byte | Resolution |
|---|---|
| 00 | 125 ms. |
| 01 | 250 ms. |
| 10 | 500 ms. |
| 11 | 1000 ms. |

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 82 |
| Command ID (LSB) | 0F |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FA |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 02 |
| Command ID (LSB) | 0F |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FA |

### 4.3.19 OS Timeout C unter (WS_OS_COUNTER)

You can use this command to query, initialize, and modify the OS timeout counter. It is used in conjunction with the OS Timeout Resolution command, described in the previous section.

At power up, this counter is disabled. When an OS Timeout Counter command is issued with non-zero data, the OS timeout counter begins to count down based on the resolution specified in the OS Timeout Resolution command. OS should reset this counter periodically with this command. If OS should miss it, the counter counts down to zero and creates an OS TIMEOUT event (0A), logs it, and transfers it to OS through the System Bus Interface and Remote Interface. This feature is executed only if the end-user starts it.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 82 |
| Command ID (LSB) | 10 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 01 |
| Data Byte 1 | (00-FF)h |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FA |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 04 |
| Type | 02 |
| Command ID (LSB) | 10 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | (00-FF)h |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 05 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | FA |

## 4.4 CPU A C ntroller C mmands

The CPU A Controller maintains the speeds for the fans on the System Board. It also maintains the LCD Display.

### 4.4.1 System Board Fans (WS_SB_FAN_HI)

You can use this command to query and modify the setting for the system fan speed. You can also use this command to set the fan speed, as shown below. This command is set if there is a fan fault, that is if WS_SB_FAN_LED is set.

| Data Byte 1 | Fan Speed |
|---|---|
| 0 | Low |
| 1 | High |

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 81 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F9 |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 01 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F9 |

### 4.4.2 System Board Fan Fault LED (WS_SB_FAN_LED)

You can use this command to query, initialize, and modify the setting for the fault LED. This command is set if a fan fault has occurred, that is if WS_SB_FANFAULT is set. An event is sent, and an entry is written to the log if the LED is on, that is if WS_SB_FAN_LED is set.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 81 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F9 |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 01 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F9 |

### 4.4.3 System Halt/Run (WS_SYS_RUN)

You can use this command to query, initialize, and modify the setting for th halt/run line. If this bit is clear, the system is halted, and an entry is written t the log. If this bit is set, the system is initialized to run, and an entry is written to the log.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 81 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F9 |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 01 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F9 |

### 4.4.4 BUS/CORE Speed Rati (WS_SB_BUSCORE)

You can use this command t query, initialize, and modify the System Board BUS/CORE speed ratio. After this command is set, the system is reset. The Reset command must be issued after the speed ratio is set, in order to change the speed ratio.. This byte is cleared when the system is powered up. This command is used primarily by the diagnostic program. Clear is the default value and this field should always be zero.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 82 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 02 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | |

### 4.4.5 System Board Fan Fault (WS_SB_FANFAULT)

You can use this command to query, initialize, and modify the settings for fan faults. If any of the bits in this byte are set, at least one fan has a fault. Bit 0 is used for fan 1, bit 2 is used for fan 2, and so on.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 82 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F9 |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 02 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F9 |

### 4.4.6 Low Limit Speed Fan (WS_SB_FAN_LOLIM)

You can use this command to query, initialize, and modify the default l w-speed setting f 1800 rpms.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 82 |
| Command ID (LSB) | 04 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F9 |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 02 |
| Command ID (LSB) | 04 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F9 |

### 4.4.7 Command Byte f r LCD Controller (WS_SB_LCD_COMMAND)

You can use this command to set up the LCD display. This byte provides low-level access to the LCD display. This is a write-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 02 |
| Command ID (LSB) | 05 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F9 |

### 4.4.8 Data Byte f r LCD C ntroller (WS_SB_LCD_DATA)

You can use this command to query, initialize, and modify the data byte for the LCD controller. This byte provides low-level access to the LCD display writing a one-character byte at the current position. This is a write-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 02 |
| Command ID (LSB) | 06 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F9 |

### 4.4.9 LCD 1 Display (WS_SB_LCD_STRING)

You can use this command is display, initialize, or modify data on the LCD controller. This command provides low-level access to the LCD Display, writing a data string at the current position. This is a write-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 03 |
| Command ID (LSB) | 07 |
| Command ID (MSB) | 00 |
| Length | 04 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F9 |

### 4.4.10 LCD First Line Display (WS_SYS_LCD1)

You can use this command to display , initialize, and modify a value on the first, or top row of a two-row LCD display, starting with the first character. This is a write-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 03 |
| Command ID (LSB) | 05 |
| Command ID (MSB) | 00 |
| Length | 14 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F9 |

### 4.4.11 LCD Sec nd Line Display (WS_SYS_LCD2)

You can use this command to display , initialize, and modify a value on the second, or bottom row of a two-row LCD display, starting with the first character. This is a write-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 03 |
| Command ID (LSB) | 06 |
| Command ID (MSB) | 00 |
| Length | 14 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F9 |

### 4.4.12 DIMM Type (WS_SB_DIMM_TYPE)

You can use this command t query the duel interface memory module (DIMM) type in each DIMM socket. This is a read-only command. Refer to NetFRAME *Wire Service Implementation User's Guide* f r m re information on DIMM types.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 83 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 10 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F9 |

### 4.4.13 Fan Speed Data (WS_SB_FAN_DATA)

You can use this command to query the System Board fan speed data, which is stored in fan number order. Approximately every second, a fan is selected and monitored with a counter for a specific length of time. The counter is loaded into the appropriate fan speed field. If the speed is not too fast, that is WS_SB_FAN_HI is clear, then it is compared to the low speed fault limit, that is, the value in WS_SB_FAN_LOLIM. If the fan speed is too slow, the appropriate bit is set in the System Board fan fault byte, that is WS_SB_FANFAULT. This is a read-only command. Refer to NetFRAME *Wire Service Implementation User's Guide* for more information on fan speeds.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 06 |
| Type | 83 |
| Command ID (LSB) | 04 |
| Command ID (MSB) | 00 |
| Length | 06 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 07 |
| Length | 06 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F9 |

## 4.5 CPU B Controller C mmands

The CPU B Controller maintains fault summaries and FRU status. It also enables the JTAG chain.

### 4.5.1 NMI Request (WS_NMI_REQ)

You can use this command to query, initialize, and modify the NMI request bit. It is a triggering pulse to the corresponding bits set in the NMI mask, that is WS_NMI_MASK. When the request bit is clear, an entry is written to the log.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 81 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 01 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | f7 |

### 4.5.2 CPU Fault Summary (WS_SB_CPU_FAULT)

You can use this command to query the CPU fault summary. First, the system determines the presence f the CPU. Then, it checks for a CPU error, the status of the CPU temperature, and the power. When this field is set, an entry is written to the log. This is a read-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 81 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

### 4.5.3 FLASH Enable (WS_SB_FLASH_ENA)

You can use this command t  query, initialize, and modify the setting for FLASH write. When this bit is set, FLASH write is enabled and the LED is turned on.

Error codes for the Status field are described in Secti n 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 81 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 01 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

### 4.5.4 FRU Status (WS_SB_FRU_FAULT)

You can use this command to query, initialize, and modify the temperature status. When the system is powered up, this bit is set. It is set or cleared by the Back Plane processors. Set is green, and clear is amber. The diagnostic program determines if the System Board, canisters, PCI slots, and CPUs are good or bad. Refer to *Maestro Recovery Manager* for more information.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 81 |
| Command ID (LSB) | 04 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 01 |
| Command ID (LSB) | 04 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

### 4.5.5 JTAG Chain Enabled (WS_SB_JTAG)

You can use this command to query, initialize, and modify the JTAG chain enable bit. When the system is powered up, this bit is clear.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 81 |
| Command ID (LSB) | 05 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 01 |
| Command ID (LSB) | 05 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

### 4.5.6 System Fault Summary (WS_SYSFAULT)

You can use this command t query, initialize, and modify the setting for the Raptor system fault summary. This bit is set if any faults are detected in the system. It is set when any of the following faults are detected:

- CPUs, that is internal error, temperature, and power at OK
- System temperature
- Canister fan faults

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 81 |
| Command ID (LSB) | 06 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 01 |
| Command ID (LSB) | 06 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

### 4.5.7 Overtemp Fault (WS_SYS_OVERTEMP)

You can use this command to query the over temperature fault. When the system is powered up, this bit is set. This is a read-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 81 |
| Command ID (LSB) | 07 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

### 4.5.8 Canister A Fan Fault (WS_CAN1_FAN_SYSFLT)

You can use this command to query, initialize, and modify the fan fault summary setting for Canister A, if present.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 81 |
| Command ID (LSB) | 08 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 01 |
| Command ID (LSB) | 08 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

### 4.5.9 Canister B Fan Fault (WS_CAN2_FAN_SYSFLT)

You can use this command to query, initialize, and modify the fan fault summary setting for Canister B, if present.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 81 |
| Command ID (LSB) | 09 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 01 |
| Command ID (LSB) | 09 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

### 4.5.10 Canister C Fan Fault (WS_CAN3_FAN_SYSFLT)

You can use this command to query, initialize, and modify the fan fault summary setting for Canister C, if present.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 81 |
| Command ID (LSB) | 0A |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 01 |
| Command ID (LSB) | 0A |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

### 4.5.11 Canister D Fan Fault (WS_CAN4_FAN_SYSFLT)

You can use this command to query, initialize, and modify the fan fault summary setting for Canister D, if present.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 81 |
| Command ID (LSB) | 0B |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 01 |
| Command ID (LSB) | 0B |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

### 4.5.12 CPU NMI Process r Mask (WS_NMI_MASK)

You can use this command t query, initialize, and modify the CPU NMI processor mask. When the system is powered up, this byte is set. Bit 0 through bit 3 represents the presence of one through four CPUs.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 82 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 02 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

### 4.5.13 CPU Error (WS_SB_CPU_ERR)

You can use this command to query the CPU error bits. Bit 0 through bit 3 represents the presence of faults for one through four CPUs. This is a read-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 82 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

### 4.5.14 CPU Power (WS_SB_CPU_POK)

Y u can use this command to query the CPU power supplies bits. Bit 0 and bit 1 represent the presence f one or two power supplies. This is a read-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 82 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

### 4.5.15 CPU Presence (WS_SB_CPU_PRES)

You can use this command to query the CPU presence bits. Bit 0 through 3 represents the presence of one through four CPUs. This is a read-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 82 |
| Command ID (LSB) | 04 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

### 4.5.16 CPU Thermal Fault (WS_SB_CPU_TEMP)

You can use this command to query the CPU thermal fault bits. This is a read-only c mmand.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 08 |
| Type | 82 |
| Command ID (LSB) | 05 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 09 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | F7 |

### 4.6 System Interface C mmands

The System Interface processor maintains the System Interface event queue.

### 4.6.1 System Interface Event Queue (WS_SI_EVENTS)

You can use this command to query, initialize, and modify the System Interface processor event bit vector. Refer to NetFRAME *Wire Service Implementation User's Guide*, for more information about events.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 20 |
| Type | 85 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 21 |
| Length | 10 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DF |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 20 |
| Type | 05 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 10 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 21 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DF |

## 4.7 Rem te Interface Commands

The Remote Interface processor maintains all of the remote port modems. It also initializes the callout script and remote events.

### 4.7.1 Remote Port Modem CD (WS_RI_CD)

You can use this command to query, initialize, and modify the setting for the remote port carrier detect (CD) modem.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 22 |
| Type | 81 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 23 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 22 |
| Type | 01 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 23 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DD |

### 4.7.2 Remote P rt Modem CTS (WS_RI_CTS)

You can use this command to query, initialize, and modify the setting for the remote port clear-to send (CTS) modem.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 22 |
| Type | 81 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 23 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 22 |
| Type | 01 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 23 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DD |

### 4.7.3 Remote P rt Modem DSR (WS_RI_DSR)

You can use this command to query, initialize, and modify the setting for the remote port data set ready (DSR) modem.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 22 |
| Type | 81 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 23 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 22 |
| Type | 01 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 23 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DD |

### 4.7.4 Remote P rt Modem DTR (WS_RI_DTR)

You can use this command t query, initialize, and modify setting for the remote port data transfer ready (DTR) modem.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 22 |
| Type | 81 |
| Command ID (LSB) | 04 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 23 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 22 |
| Type | 01 |
| Command ID (LSB) | 04 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 23 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DD |

### 4.7.5 Remote P rt Modem RTS (WS_RI_RTS)

You can use this command to query, initialize, and modify the setting for the remote port request to send (RTS) modem.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes  n page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 22 |
| Type | 81 |
| Command ID (LSB) | 05 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 23 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 22 |
| Type | 01 |
| Command ID (LSB) | 05 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 23 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DD |

### 4.7.6 Call ut Script Initializati n (WS_RI_CALLOUT)

You can use this command to invoke a callout script, which is programmed in WS_SYS_CALL_SCRIPT. A value is passed as an argument to the script. An entry is written t the log. The format has not been determined at this time.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 22 |
| Type | 82 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 23 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 22 |
| Type | 02 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 23 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DD |

### 4.7.7 Rem te Interface Event Queue (WS_RI_EVENTS)

You can use this command to query, initialize, and modify the System Interface event bit vector. Refer to NetFRAME *Wire Service Implementation User's Guide*, for m re information about events.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 22 |
| Type | 85 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 02 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 23 |
| Length | 02 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DD |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 22 |
| Type | 05 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 02 |
| Data Byte 1 | |
| : | |
| Data Byte | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 23 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | DD |

## 4.8 Canister Controller C mmands

The Canister Controller processor maintains fan speeds and faults.

There are four occurrences of the commands in this processor. Most of the fields are the same for all occurrences, so only Canister A is documented. However, the **Slave Addr** fields are different for each occurrence, and are listed below:

| Canister No. | Request Slave Addr | Response Slave Addr | Inverted Slave Addr |
|---|---|---|---|
| A | 40 | 41 | BF |
| B | 42 | 43 | BD |
| C | 44 | 45 | BB |
| D | 46 | 47 | B9 |

### 4.8.1 Canister Fan High Speed (WS_CAN_FAN_HI)

You can use this command to query, initialize, and modify the canister fan high speed setting. If the speed is too high, that is WS_SB_FAN_LED is set, this bit is set.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 40 |
| Type | 81 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 41 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | BF |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 40 |
| Type | 01 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 41 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | BF |

### 4.8.2 Canister Fan Fault LED (WS_CAN_FAN_LED)

You can use this command to query, initialize, and modify the canister fan fault LED setting. This bit is set if the canister has a fault, that is WS_CAN_FANFAULT1 is set. An entry is written to the log when this bit is set.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 40 |
| Type | 81 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 41 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | BF |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 40 |
| Type | 01 |
| Command ID (LSB) | 02 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 41 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | BF |

### 4.8.3 Canister JTAG TMS Chain (WS_CAN_JTAG_ENA)

(NOT USED)

### 4.8.4 Card in Slot 5 (WS_CAN_NMI_S5)

(NOT USED)

### 4.8.5 Canister PCI Slot Power (WS_CAN_POWER)

You can use this command to query, initialize, and modify the setting for the PCI slot power in the canister. There is a small (about 5 ms) delay between each set. An entry is written to the log when this bit is set or cleared.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 40 |
| Type | 81 |
| Command ID (LSB) | 05 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 41 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | BF |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 40 |
| Type | 01 |
| Command ID (LSB) | 05 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 41 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | BF |

### 4.8.6 Slot 5 Presence Indicat r (WS_CAN_S5_PRESENT)

(NOT USED)

### 4.8.7 Passive Board N t in Sl t 5 (WS_CAN_S5_SMART)

(NOT USED)

### 4.8.8 Canister Fan Low Speed Fault Limit (WS_CAN_FAN_LOLIM)

You can use this command to query, initialize, and modify the setting for the canister fan low speed fault limit. It is set to the equivalent of 1000 rpms when the system is powered up.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 40 |
| Type | 82 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 41 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | BE |

**WRITE**

Request:

| Field | Value |
|---|---|
| Slave Addr | 40 |
| Type | 02 |
| Command ID (LSB) | 01 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Data Byte 1 | |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 41 |
| Length | 00 |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | BF |

### 4.8.9 Presence f PCI Card Sl t (WS_CAN_PCI_PRESENT)

(NOT USED)

### 4.8.10 Canister Fan Fault (WS_CAN_FANFAULT)

You can use this command to query the canister fan fault status. This is a read-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 40 |
| Type | 81 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 41 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | BF |

### 4.8.11 Canister Fan Speed Data (WS_CAN_FAN_DATA)

You can use this command to query the canister fan speed. Approximately every second a fan is selected and monitored, for specific length of time. The counter is then loaded into the appropriate fan speed. If the speed is not too fast, that is WS_CAN_FAN_HI is clear, then the speed is compared to the slow speed limit, that is WS_CAN_FAN_LOLIM is set. If the fan speed is too slow, the appropriate bit is set in the fan fault byte, that is WS_CAN_FANFAULT. If the speed is not too slow, this field is clear. This is a read-only command.

Error codes for the Status field are described in Section 3.2 Wire Service Status Codes on page 7.

**READ**

Request:

| Field | Value |
|---|---|
| Slave Addr | 40 |
| Type | 83 |
| Command ID (LSB) | 03 |
| Command ID (MSB) | 00 |
| Length | 01 |
| Check Sum | |

Response:

| Field | Value |
|---|---|
| Slave Addr | 41 |
| Length | 01 |
| Data Byte 1 | |
| Status | 00 |
| Check Sum | |
| Inverted Slave Addr | BF |

"InterPulse" spec
call Wire Service

notes - see p.28 — Issues needing resolution — take out a fill in.

p.29 remote interface and call out script syntax not described

Need Arch. Drawing i.e. ServerArch at p.
of some
"Fans"
"Thermo"
whatever else
"Hotswap"

# Raptor Wire Service Architecture

Version 1.0

1/23/96

Prepared for
Raptor Implementation Group

by
Karl Johnson (KJ)

# Table of Contents

# Introduction

"Wire Service" is the code name for the Raptor project system control, diagnostic and maintenance bus ( formerly known as the CDM bus). Raptor is a completely "fly by wire" system - no switch, indicator or other control is directly connected to the function it monitors or controls, instead all the control and monitoring connections are made by the network of processors that comprise the "Wire Service" for the system. The processors are Microchip PIC processors and the network is a 400 kbps I²C serial bus. A limited understanding of I²C protocol is a prerequisite for understanding Wire Service protocols (See "The I²C-bus and how to use it" - Philips Semiconductor, Jan 1992). Control on this bus is distributed, each processor can be either a master or a slave and can control resources on itself or any other processor on the bus

# Logical Model

All of the command, diagnostic, monitoring and history functions are accessed using a global network memory model. That is, any function may be queried simply by generating a network "read" request targeted at the function's known global network address. In the same fashion, a function may be exercised simply by "writing" to its global network address. Any Wire Service processor may initiate read/write activity by sending a message on the I²C bus to the processor responsible for the function ( which can be determined from the known global address of the function ). The network memory model includes typing information as part of the memory addressing information. It implements separate address spaces for each data type. This allows for compact internal storage and creating of unique more complex data types better suited to specific functions.

# Message Protocol

Using a network global memory model places relatively modest requirements for the I²C message protocol.

- All messages conform to the I²C message format including addressing and read/write indication.
- All I²C messages use 7 bit addressing and the "General Call" address is not used.
- Any processor can originate (be a Master) or respond (be a Slave)
- All message transactions consist of I²C "Combined format" messages. This is made up of two back to back I²C simple messages with a repeated START condition between (which does not allow for re-arbitrating the bus). The first message is always a Write ( Master to Slave ) and the second message is a Read ( Slave to Master).
- Only two types of transactions are used: Memory-Read and Memory-Write.
- Sub-Addressing formats vary depending on data type being used.

## Generic Wire Service I$^2$C Message Format

The generic Wire Service message format is designed for easy encode/decode and reliability. It is not necessarily always the most compact representation possible

Master Asserts START

| Offset | MSB | LSB | |
|---|---|---|---|
| Byte 0 | Slave Address | 0 | 0 means I$^2$C write to slave |
| Byte 1 | Data Type | R/W | R/W = 0 Mem Write / 1 Mem Read |
| Byte 2 | Sub-Address | | Most Significant Byte of Address |
| Byte 3 | Sub-Address ( Continued ) | | Least Significant Byte of Address |
| Byte 4 | Length of Data | | Length write or read buff size |
| Byte 5 | Data | | Present only for memory write |
| : | : | | : |
| Byte N | Checksum | | |

Master repeats START

| | | | |
|---|---|---|---|
| Byte 0 | Slave Address (same) | 1 | 1 means I$^2$C read from slave |
| Byte 1 | Length of Data | | Length of Data only (N-3) |
| Byte 2 | Data | | |
| : | : | | |
| Byte N-1 | Status | | Status 0=Success otherwise Failure |
| Byte N | Checksum | | |

# Data Type Descriptions

Each data type description includes a rational for its existence and an example simple protocol message.

## Bit Type

The bit data type is to be used for simple logic valued items (TRUE/FALSE, ON/OFF, etc.)

Read Bit Message

Request

| | | | | | |
|---|---|---|---|---|---|
| Slave Address | Type/RW | Bit Addr MSB | Bit Addr LSB | Request<br>1 | Check Byte |

Response

| | | | | |
|---|---|---|---|---|
| Slave Address | Length<br>1 | Bit Value<br>(0/1) | Status<br>0/Success | Check Byte |

Write Bit Message

Request

| | | | | | | |
|---|---|---|---|---|---|---|
| Slave Address | Type/RW | Bit Addr MSB | Bit Addr LSB | Length<br>1 | Bit Value<br>(0/1) | Check Byte |

Response

| | | | |
|---|---|---|---|
| Slave Address | Length<br>0 | Status<br>0/Success | Check Byte |

## Byte Type

The byte data type is to be used for single byte valued items (0-255)

Read Byte Message

Request

| | | | | | |
|---|---|---|---|---|---|
| Slave Address | Type/RW | Byte Addr MSB | Byte Addr LSB | Request<br>1 | Check Byte |

Response

| | | | | |
|---|---|---|---|---|
| Slave Address | Length<br>1 | Byte Value<br>(0-255) | Status<br>0/Success | Check Byte |

Write Byte Message

Request

| | | | | | | |
|---|---|---|---|---|---|---|
| Slave Address | Type/RW | Byte Addr MSB | Byte Addr LSB | Length 1 | Byte Value (0-255) | Check Byte |

Response

| | | | |
|---|---|---|---|
| Slave Address | Length 0 | Status 0/Success | Check Byte |

## String Type

The String type is designed to handle data that is best organized as variable length strings of 0 to 255 bytes. Internal allocation of string storage for each string may be less than 255 bytes, and writing a string longer than available storage will return an error.

Read String Message

Request

| | | | | | |
|---|---|---|---|---|---|
| Slave Address | Type/RW | String Addr MSB | String Addr LSB | Request 0-255 | Check Byte |

Response

| | | | | | | |
|---|---|---|---|---|---|---|
| Slave Address | Length N | String Data 1 | . . . | String Data N | Status 0/Success | Check Byte |

Write String Message

Request

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Slave Address | Type/RW | String Addr MSB | String Addr LSB | Length N | String Data 0 | . . . | String Data N | Check Byte |

Response

| | | | |
|---|---|---|---|
| Slave Address | Length 0 | Status 0/Success | Check Byte |

## Lock Byte Type

The lock byte data type is to be used for single bytes of data used to control synchronization as it performs a test and set operation on the byte. If the byte is zero it is set to one and the original value of the byte is returned. This data type shares the same address space with the Byte Array data type.

Lock Byte Message (Read Type)

Request

| Slave Address | Type/RW | Byte Addr MSB | Byte Addr LSB | Request 1 | Check Byte |
|---|---|---|---|---|---|

Response

| Slave Address | Length 1 | Original Byte Value | Status 0/Success | Check Byte |
|---|---|---|---|---|

## Byte Array Type

The byte array data type is to be used for general storage of data that is unanticipated in this architecture. This storage is only implemented in the Wire Service processor with NVRAM. External code will be responsible for managing allocation/deallocation and data directory information. Recommend that byte at address 0 be the lock byte for modification of the data directory.

Read Byte Array Message

Request

| Slave Address | Type/RW | Start Addr MSB | Start Addr LSB | Request 1-255 | Check Byte |
|---|---|---|---|---|---|

Response

| Slave Address | Length N | Array Data 1 | . . . | Array Data N | Status 0/Success | Check Byte |
|---|---|---|---|---|---|---|

Write Byte Array Message
Request

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Slave Address | Type/RW | Start Addr MSB | Start Addr LSB | Length N | Array Data 1 | . . . | Array Data N |
| Check Byte | | | | | | | |

Response

| | | | |
|---|---|---|---|
| Slave Address | Length 0 | Status 0/Success | Check Byte |

## Log Type

The Log data type is to be used for logging byte strings in circular log buffer. It is used to record system events in the NVRAM system log.

Read Log Message
Request

| | | | | | |
|---|---|---|---|---|---|
| Slave Address | Type/RW | Log Addr MSB | Log Addr LSB | Request 255 | Check Byte |

Response

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Slave Address | Length N+7 | Log Time MSB | Log Time | Log Time | Log Time LSB | Log Addr MSB | Log Addr LSB |
| Log Data Byte 0 | . . . | Log Data Byte N | Status 0/Success | Check Byte | | | |

Write Log Message
Request

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Slave Address | Type/RW | N/A | N/A | Length N | Log Data Byte 0 | . . . | Log Data Byte N |
| Check Byte | | | | | | | |

Response

| | | | |
|---|---|---|---|
| Slave Address | Length 0 | Status 0/Success | Check Byte |

The addressing of log entries has some special characteristics.
1) Reading address 65565 (0xffff) is special - It represents the address of the latest entry in the log.
2) Reading address 65564 (0xfffe) is also special - It represents the address of the earliest available entry.

3) The address of real log entries wraps at 65519 (0xffef). The next sequential entry after 65519 is 0.
4) The address of is ignored on write and the next available entry is written.
5) To read the entire log in forward time order, read entry at address 65564. This returns the first log entry along with its actual log address. Increment that address by one and read that entry. Repeat the last step until status indicates failure.
6) To read the entire log in reverse time order, read entry at address 65565. This returns the last log entry along with its actual log address. Decrement that address by one and read that entry. Repeat the last step until status indicates failure.
7)To keep a complete external copy of the log, first read the entire log in forward time order and remember the last valid entry. Then periodically read forward from the remembered last valid entry to the end and add that to the external copy.

## Event Type

The event data type is to be used for alerting external interfaces of events in the Wire Service network. Event memory is organized as a queue. The queue will probably be quite small (< 20 Events). Writing an event places the event ID at the next available entry, unless the last queue entry would be written by this event. In that case, the last queue entry is a Queue Overflow Event and the write fails. This allows the external interface to realize that events were lost and it should scan for any changes in data.
Reading the event type returns requested number of events in the queue or the entire queue which ever is less and removes them from the queue.

Read Event Message
Request

| | | | | | |
|---|---|---|---|---|---|
| Slave Address | Type/RW | N/A | N/A | Request 1-255 | Check Byte |

Response

| | | | | | | |
|---|---|---|---|---|---|---|
| Slave Address | Length N | Event ID 1 | ... | Event ID N | Status 0/Success | Check Byte |

Write Event Message
Request

| | | | | | | |
|---|---|---|---|---|---|---|
| Slave Address | Type/RW | N/A | N/A | Length<br>1 | Event ID | Check Byte |

Response

| | | | |
|---|---|---|---|
| Slave Address | Length<br>0 | Status<br>0/Success | Check Byte |

Possible Event Types:
CPU Status Change
Power Status Change
Canister Status Change
Fan Status Change

## Screen Type

The screen data type is to be used for communication of character mode screen information from the system BIOS to remote management interface.

Read Screen Message
Request

| | | | | | |
|---|---|---|---|---|---|
| Slave Address | Type/RW | S Addr MSB | S Addr LSB | Request<br>1-255 | Check Byte |

Response

| | | | | | | |
|---|---|---|---|---|---|---|
| Slave Address | Length<br>N | Screen Data 1 | . . . | Screen Data N | Status<br>0/Success | Check Byte |

Write Screen Message
Request

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Slave Address | Type/RW | S Addr MSB | S Addr LSB | Length<br>N | Screen Data 1 | . . . | Screen Data n |
| Check Byte | | | | | | | |

Response

| | | | |
|---|---|---|---|
| Slave Address | Length<br>0 | Status<br>0/Success | Check Byte |

The screen address space consists of an image of character video memory for a 80x50 screen. Each character cell has both a character byte and attribute byte for a total of 8000 bytes of screen memory. Additionally memory is implemented up to address 8191. Use of the bytes above 8000 are defined by the BIOS and the remote managment software. Addresses 8001 and 8002 are suggested as the cursor address register.

## Queue Type

The Queue type is designed to move data between the system and a remote management software. Queue elements are variable length up to 255 bytes. If there is no room to add an element to a queue, the entry is not added and failure status is returned. If there is no queue element available to read, failure status is returned. Interpretation of data in the queue elements is left to external software.

Read Queue Message

Request

| | | | | | |
|---|---|---|---|---|---|
| Slave Address | Type/RW | N/A | N/A | Request 0-255 | Check Byte |

Response

| | | | | | | |
|---|---|---|---|---|---|---|
| Slave Address | Length N | Queue Data 1 | . . . | Queue Data N | Status 0/Success | Check Byte |

Write Queue Message

Request

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Slave Address | Type/RW | N/A | N/A | Length N | Queue Data 0 | . . . | Queue Data N | Check Byte |

Response

| | | | |
|---|---|---|---|
| Slave Address | Length 0 | Status 0/Success | Check Byte |

# System Bus Interface

## Introduction

The system Bus interface to the Wire Service I$^2$C bus is implemented by a dedicated Wire Service System Interface Processor (WSSIP)and 2 message FIFOs, one for message data written to the system bus interface (requests) and one for message data returned by the system bus interface (responses). The system bus interface appears in system I/O space as 2 registers, each 8 bits wide. The lower register is the message data register (MDR) and the upper register is the command and status register (CSR).

The MDR, when written from the system bus, loads a byte into the request FIFO and when read from the system bus presents a byte unloaded from the response FIFO if any. The FIFOs may be most quickly loaded and unloaded via REP OUT or REP IN instructions executed on the system processor(s). The protocol is designed to load and unload these FIFOs with a minimum of interpretation of the message data (i.e. all message fragments have length fields in the same position and same interpretation). These FIFOs serve two functions: 1) They match speeds between the very fast System Bus and the slower WSSIP and I$^2$C bus and 2) They temporarily serve as interim memory for the messages relieving the Wire Service processor of that requirement.

The CSR is implemented with a direct interface to the WSSIP. Writing to this register from the system bus interrupts the WSSIP and may cause it to take actions depending on the value written. One action could be to change the value returned if there is a read from the system bus to this register. Note that there is no hardwired relation between values written to and read from this register and also that there is a processing interval between writing from the system bus to this register and the WSSIP reading the register value written. This could lead to data loss should the system bus write a second time to the CSR before the WSSIP read the first item. To avoid this problem, all writes to the CSR eventually result in some change to the value read from the CSR. Thus after writing the CSR, you should wait to see the change in the CSR indicating it has processed the data written to it.

## Interface Operation

Operation of the system bus interface is controlled through the CSR.

CSR Write Format

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| Int Req | Command Value | | | | | | |

Field Descriptions
Command Value - The value corresponding to the command to execute.
Int Req - Commands are modified by this bit to request an interrupt upon completion.

CSR Read Format

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| Int Pend | Int Ena | Events | | | | Done | Allocated |

Field Descriptions

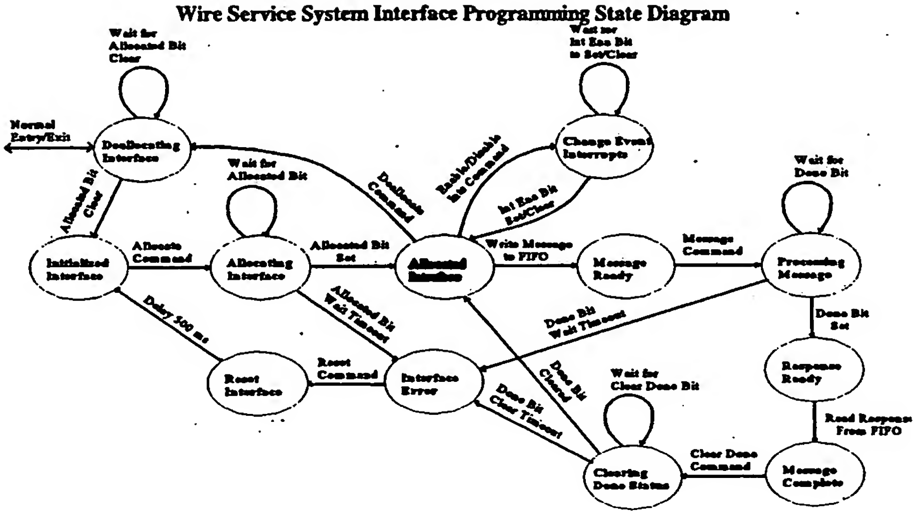
| | |
|---|---|
| Allocated | - The interface is currently in use. Only Allocate commands will be accepted until set. |
| Done | - A command has completed ( except for Allocate, Deallocate and Clear Done ). |
| Events | - The Event Queue on the WSSIP is not empty if this bit is set. |
| Int Ena | - Interrupt on Events bit going from 0 to 1 is enabled. |
| Int Pending | - An interrupt source is active. |

CSR commands that are currently defined and their actions:

| Commands | Value | Actions |
|---|---|---|
| Allocate | 01<br>0x01 | Clears both FIFOs of any stale data, clears Done and then sets Allocated. Generally, it should be first command of any transaction sequence. |
| Deallocate | 02<br>0x02 | Clears Done and Allocated. Generally the last command of transaction sequence. |
| Enable Ints | 03<br>0x03 | Enables interrupts on Events bit being set. Sets Done. May cause interrupt if Events is already set. |
| Disable Ints | 04<br>0x04 | Disables interrupts on Events bit being set. Sets Done. |
| Message | 05<br>0x05 | Request in request FIFO is sent on the I$^2$C bus and the response is received into response FIFO. Sets Done. I$^2$C bus errors create unique error response. |
| Clear Done | 06<br>0x06 | Clears the Done bit. |
| Reset | 165<br>0xa5 | Unconditionally clears all bits in the Read CSR except "Events". Aborts any currently in progress message operation and clears any interrupt. |

The following is a state diagram that details the expected use of the Wire Service system interface.

## Wire Service System Interface Programming State Diagram



Not every possible error condition and possible state transition is anticipated by the above diagram, but it does indicate the intended operational sequence to be used with the interface in non-interrupt driven mode. To extend the diagram to interrupt mode, simply add the interrupt request bit to each command desired and instead of waiting for the appropriate bit, exit and wait for the interrupt. In most cases, the only command in which it makes sense to request an interrupt is the Message command. All other commands either complete very quickly or ignore the interrupt bit (Reset Command).

Note: The Reset Command is a problem determining if it is complete as it sets no bit and may take some time to work because of existing activity, so it seems the only way to do it now it to set it and wait.

## Wire Service Network Physical Connections

The following table describe all of the physical signal connections to all of the Wire Service processors. The names for the connections will be related to network accessible memory data in the section which follows called "Wire Service Network Memory Map".

Note: All signal types and definitions are from the viewpoint of the individual Wire Service PIC processor ( e.g. Input means input to PIC processor )

Wire Service System Bus Interface (System Type ID: S0) Processor ID 10

| Pin | Type | Name | Function | Notes |
|---|---|---|---|---|
| RA0 | I | S0_FIFO_IEFZ | In FIFO (ISA Writes) Empty Flag (Active Low) | Status Flags for both ISA bus FIFO's.<br>Need to monitor for incoming message overruns. Also must assure that output FIFO is empty before loading output message. |
| RA1 | I | S0_FIFO_IHFZ | In FIFO (ISA Writes) Half-full Flag (Active Low) | |
| RA2 | I | S0_FIFO_IFFZ | In FIFO (ISA Writes) Full Flag (Active Low) | |
| RA3 | I | S0_FIFO_OEFZ | Out FIFO (ISA Reads) Empty Flag (Active Low) | |
| RA4 | I | S0_FIFO_OHFZ | Out FIFO (ISA Reads) Half-full Flag (Active Low) | |
| RA5 | I | S0_FIFO_OFFZ | Out FIFO (ISA Reads) Full Flag (Active Low) | |
| RB0 | I/O | S0_FIFO_D0 | ISA FIFOs Data bus Bit 0 | This is an 8 bit bi-directional port for the ISA FIFO data bus. These bits should drive the bus before asserting FIFO Write and can be read after tri-stating and asserting FIFO Read |
| RB1 | I/O | S0_FIFO_D1 | ISA FIFOs Data bus Bit 1 | |
| RB2 | I/O | S0_FIFO_D2 | ISA FIFOs Data bus Bit 2 | |
| RB3 | I/O | S0_FIFO_D3 | ISA FIFOs Data bus Bit 3 | |
| RB4 | I/O | S0_FIFO_D4 | ISA FIFOs Data bus Bit 4 | |
| RB5 | I/O | S0_FIFO_D5 | ISA FIFOs Data bus Bit 5 | |
| RB6 | I/O | S0_FIFO_D6 | ISA FIFOs Data bus Bit 6 | |
| RB7 | I/O | S0_FIFO_D7 | ISA FIFOs Data bus Bit 7 | |
| RC0 | O | S0_FIFO_RZ | FIFO (ISA Writes) Read (Assert Low) | Assert to read the In FIFO |
| RC1 | O | S0_FIFO_WZ | PIC to ISA FIFO (ISA Reads) Write (Assert Low) | Assert to write the Out FIFO |
| RC2 | O | S0_ISA_INT | PIC to ISA Interrupt Request (Assert ? ) | Level Interrupt to ISA bus |
| RC3 | I/O | S0_I2C_DATA | Wire Service Bus Clock (I2C) | Only used for I2C |
| RC4 | I/O | S0_I2C_CLK | Wire Service Bus Data (I2C) | |
| RC5 | O | S0_FIFO_RSTZ | In and Out FIFOs Reset (Assert Low) | Resets both FIFOs |
| RC6 | O | S0_FIFO_IRTZ | In FIFO (ISA Writes) Retransmit (Assert Low) | Resets the Read pointer to 0 |
| RC7 | O | S0_FIFO_ORTZ | Out FIFO (ISA Reads) Retransmit (Assert Low) | Resets the Read pointer to 0 |
| RD0 | I/O | S0_CSR_D0 | ISA External Data bus Bit 0 ( Slave parallel port ) | The slave parallel port is used as a bidirectional control and status register on the ISA bus. |
| RD1 | I/O | S0_CSR_D1 | ISA External Data bus Bit 1 ( Slave parallel port ) | |
| RD2 | I/O | S0_CSR_D2 | ISA External Data bus Bit 2 ( Slave parallel port ) | |
| RD3 | I/O | S0_CSR_D3 | ISA External Data bus Bit 3 ( Slave parallel port ) | |
| RD4 | I/O | S0_CSR_D4 | ISA External Data bus Bit 4 ( Slave parallel port ) | |
| RD5 | I/O | S0_CSR_D5 | ISA External Data bus Bit 5 ( Slave parallel port ) | |
| RD6 | I/O | S0_CSR_D6 | ISA External Data bus Bit 6 ( Slave parallel port ) | |
| RD7 | I/O | S0_CSR_D7 | ISA External Data bus Bit 7 ( Slave parallel port ) | |
| RE0 | I | S0_CSR_RZ | ISA Read Slave Parallel Port (Assert Low) | Not directly manipulated after setting port D/E to act as slave parallel port |
| RE1 | I | S0_CSR_WZ | ISA Write Slave Parallel Port (Assert Low) | |
| RE2 | I | S0_CSR_SZ | ISA Slave Parallel Port Select (Assert Low) | |

Wire Service System Monitor A (System Type ID S1) Processor ID 3

| Pin | Type | Name | Function | Notes |
|---|---|---|---|---|
| RA0 | O | S1_FAN_HI | System Board fan speed to high (Assert Hi) | Assert on any SB fan failure |
| RA1 | O | S1_SBFAN_LED | System Board fan fault LED | Assert on any SB fan failure |
| RA2 | O | S1_BC_DS0 | Bus/Core Speed Ratio and DIMM Select Mux Bit 0 | During system reset these bits select bus/core speed ratio for all processors. Otherwise they select which DIMM presents its type on DIMM type port. |
| RA3 | O | S1_BC_DS1 | Bus/Core Speed Ratio and DIMM Select Mux Bit 1 | |
| RA4 | O | S1_BC_DS2 | Bus/Core Speed Ratio and DIMM Select Mux Bit 2 | |
| RA5 | O | S1_BC_DS3 | Bus/Core Speed Ratio and DIMM Select Mux Bit 3 | |
| RB0 | I/O | S1_LCD_D0 | LCD Controller Data Bus bit 0 | These lines make up the 8 bit data bus to the LCD display |
| RB1 | I/O | S1_LCD_D1 | LCD Controller Data Bus bit 1 | |
| RB2 | I/O | S1_LCD_D2 | LCD Controller Data Bus bit 2 | |
| RB3 | I/O | S1_LCD_D3 | LCD Controller Data Bus bit 3 | |
| RB4 | I/O | S1_LCD_D4 | LCD Controller Data Bus bit 4 | |
| RB5 | I/O | S1_LCD_D5 | LCD Controller Data Bus bit 5 | |
| RB6 | I/O | S1_LCD_D6 | LCD Controller Data Bus bit 6 | |
| RB7 | I/O | S1_LCD_D7 | LCD Controller Data Bus bit 7 | |
| RC0 | I | S1_FAN_TP | Tachometer pulse input from selected fan | Generally routed to counter |
| RC1 | O? | S1_OK_TO_RUN | Drives SYS_PWRGOOD signal | System starts on 0->1 transition |
| RC2 | I | S1_RESET_SW | Undebounced input from System Reset switch | |
| RC3 | I/O | S1_I2C_DATA | Wire Service Bus Clock (I2C) | Only used for I2C |
| RC4 | I/O | S1_I2C_CLK | Wire Service Bus Data (I2C) | |
| RC5 | O | S1_FAN_SEL0 | Fan Tachometer Multiplexer Select Bit 0 | Used to select which fan tachometer pulse output is gated to S1_FAN_TP |
| RC6 | O | S1_FAN_SEL1 | Fan Tachometer Multiplexer Select Bit 1 | |
| RC7 | O | S1_FAN_SEL2 | Fan Tachometer Multiplexer Select Bit 2 | |
| RD0 | I | S1_DIMM_D0 | DIMM Type port bit 0 | These lines make up an 8 bit port which on which the DIMM module in the slot selected by S1_BC_DS0..3 presents its type data if any. If no DIMM is present in the slot selected the DIMM type bits are all 1's. |
| RD1 | I | S1_DIMM_D1 | DIMM Type port bit 1 | |
| RD2 | I | S1_DIMM_D2 | DIMM Type port bit 2 | |
| RD3 | I | S1_DIMM_D3 | DIMM Type port bit 3 | |
| RD4 | I | S1_DIMM_D4 | DIMM Type port bit 4 | |
| RD5 | I | S1_DIMM_D5 | DIMM Type port bit 5 | |
| RD6 | I | S1_DIMM_D6 | DIMM Type port bit 6 | |
| RD7 | I | S1_DIMM_D7 | DIMM Type port bit 7 | |
| RE0 | O | S1_LCD_RS | LCD Controller Register Select | See LCD Controller data sheet for details of operation of these signals |
| RE1 | O | S1_LCD_ENA | LCD Controller Register Enable | |
| RE2 | O | S1_LCD_RW | LCD Controller Register Read/Write | |

Wire Service System Monitor B (System Type ID S2) Processor ID 4

| Pin | Type | Name | Function | Notes |
|---|---|---|---|---|
| RA0 | O | S2_FLASH_LED | CPU Display the Enable/Disable state of the BIOS Flash ROM | Should track S2_FLASH_ENA |
| RA1 | O | S2_SBFLT_LED0 | System Board FRU LED Pin 0 (bicolor LED) | Drive in different combinations for OFF, AMBER, GREEN |
| RA2 | O | S2_SBFLT_LED1 | System Board FRU LED Pin 1 (bicolor LED) | |
| RA3 | O | S2_OVRTMP_LED | Over Temperature LED | |
| RA4 | I | S2_TEMP_CPU4 | Thermal Fault - CPU 4 | Indicator that CPU has exceeded temperature limit and faulted |
| RA5 | I | S2_TEMP_CPU3 | Thermal Fault - CPU 3 | |
| RB0 | O | S2_SB_JTAG | Enable System Board JTAG Chain TMS | |
| RB1 | O | S2_FLASH_WE | System BIOS FLASH Write Enable | |
| RB2 | I | S2_FLASH_SW | System BIOS FLASH Write Enable Switch ( undebounced ) | |
| RB3 | I | S2_NMI_SW | System Non-Maskable Interrupt (NMI) Switch (undebounced) | |
| RB4 | I | S2_POK_CPU1 | Power Good signal from CPU 1 | Indicator that power regulator for CPU is operating correctly. Only valid if corresponding CPU is present (S2_PRES_CPUx) |
| RB5 | I | S2_POK_CPU2 | Power Good signal from CPU 2 | |
| RB6 | I | S2_POK_CPU3 | Power Good signal from CPU 3 | |
| RB7 | I | S2_POK_CPU4 | Power Good signal from CPU 4 | |
| RC0 | x | | Unused | |
| RC1 | x | | Unused | |
| RC2 | O | S2_NMI_CPU4 | NMI Request for CPU 4 | Toggle to cause NMI to CPU |
| RC3 | I/O | S2_I2C_CLK | Wire Service Bus Data (I2C) | Only used for I2C |
| RC4 | I/O | S2_I2C_CLK | Wire Service Bus Data (I2C) | |
| RC5 | O | S2_NMI_CPU3 | NMI Request for CPU 3 | See S2_NMI_CPU4 above |
| RC6 | O | S2_NMI_CPU2 | NMI Request for CPU 2 | |
| RC7 | O | S2_NMI_CPU1 | NMI Request for CPU 1 | |
| RD0 | I | S2_PRES_CPU1 | Presence detection bit - CPU 1 | Asserted when a processor inserted in the system board |
| RD1 | I | S2_PRES_CPU2 | Presence detection bit - CPU 2 | |
| RD2 | I | S2_PRES_CPU3 | Presence detection bit - CPU 3 | |
| RD3 | I | S2_PRES_CPU4 | Presence detection bit - CPU 4 | |
| RD4 | I | S2_ERROR_CPU1 | Processor Fault bit - CPU 1 | Processor either failed BIST on startup or later other fault. Only valid if corresponding CPU is present (S2_PRES_CPUx) |
| RD5 | I | S2_ERROR_CPU2 | Processor Fault bit - CPU 2 | |
| RD6 | I | S2_ERROR_CPU3 | Processor Fault bit - CPU 3 | |
| RD7 | I | S2_ERROR_CPU4 | Processor Fault bit - CPU 4 | |
| RE0 | O | S2_SYSFLT_LED | System Fault summary LED | |
| RE1 | I | S2_TEMP_CPU2 | Thermal Fault - CPU 2 | See S2_TEMP_CPU4 above |
| RE2 | I | S2_TEMP_CPU1 | Thermal Fault - CPU 1 | |

Wire Service System Recorder (System Type ID S3 ) Processor ID 1

| Pin | Type | Name | Function | Notes |
|---|---|---|---|---|
| RA0 | O | S3_NVRAM_A8 | NVRAM Address Bit 8 | NVRAM Address Bus Bits 8-13 |
| RA1 | O | S3_NVRAM_A9 | NVRAM Address Bit 9 | |
| RA2 | O | S3_NVRAM_A10 | NVRAM Address Bit 10 | |
| RA3 | O | S3_NVRAM_A11 | NVRAM Address Bit 11 | |
| RA4 | O | S3_NVRAM_A12 | NVRAM Address Bit 12 | |
| RA5 | O | S3_NVRAM_A13 | NVRAM Address Bit 13 | |
| RB0 | I/O | S3_NVRAM_D0 | NVRAM Data Bit 0 | NVRAM 8 Bit Data Bus |
| RB1 | I/O | S3_NVRAM_D1 | NVRAM Data Bit 1 | |
| RB2 | I/O | S3_NVRAM_D2 | NVRAM Data Bit 2 | |
| RB3 | I/O | S3_NVRAM_D3 | NVRAM Data Bit 3 | |
| RB4 | I/O | S3_NVRAM_D4 | NVRAM Data Bit 4 | |
| RB5 | I/O | S3_NVRAM_D5 | NVRAM Data Bit 5 | |
| RB6 | I/O | S3_NVRAM_D6 | NVRAM Data Bit 6 | |
| RB7 | I/O | S3_NVRAM_D7 | NVRAM Data Bit 7 | |
| RC0 | O | S3_NVRAM_CSZ | NVRAM Chip Select (Negative Logic) | Control signals for NVRAM - See Dallas DS1245 data sheet |
| RC1 | O | S3_NVRAM_OEZ | NVRAM Output Enable (Negative Logic) | |
| RC2 | O | S3_NVRAM_WEZ | NVRAM Write Enable (Negative Logic) | |
| RC3 | I/O | S3_I2C_CLK | Wire Service Bus Data (I2C) | Only used for I2C |
| RC4 | I/O | S3_I2C_CLK | Wire Service Bus Data (I2C) | |
| RC5 | O | S3_NVRAM_A14 | NVRAM Address Bit 14 | NVRAM Address Bus Bits 14-16 |
| RC6 | O | S3_NVRAM_A15 | NVRAM Address Bit 15 | |
| RC7 | O | S3_NVRAM_A16 | NVRAM Address Bit 16 | |
| RD0 | O | S3_NVRAM_A0 | NVRAM Address Bit 0 | NVRAM Address Bus Bits 0-7 |
| RD1 | O | S3_NVRAM_A1 | NVRAM Address Bit 1 | |
| RD2 | O | S3_NVRAM_A2 | NVRAM Address Bit 2 | |
| RD3 | O | S3_NVRAM_A3 | NVRAM Address Bit 3 | |
| RD4 | O | S3_NVRAM_A4 | NVRAM Address Bit 4 | |
| RD5 | O | S3_NVRAM_A5 | NVRAM Address Bit 5 | |
| RD6 | O | S3_NVRAM_A6 | NVRAM Address Bit 6 | |
| RD7 | O | S3_NVRAM_A7 | NVRAM Address Bit 7 | |
| RE0 | O | S3_RTC_CLK | Real Time Clock - Data Clock | See Dallas DS1603 data sheet |
| RE1 | I | S3_RTC_DATA | Real Time Clock - Serial Data | |
| RE2 | O | S3_RTC_RSTZ | Real Time Clock - Protocol Reset (Negative Logic) | |

Wire Service Backplane (System Type ID S4) Processor ID 2

| Pin | Type | Name | Function | Notes |
|---|---|---|---|---|
| RA0 | A | S4_VOLTS_P5V | Analog measure of system +5 volt main supply | Use D/A converter to read voltages as 0-255. Calibration constants are determined externally |
| RA1 | A | S4_VOLTS_P3V | Analog measure of system +3.3 volt main supply | |
| RA2 | A | S4_VOLTS_P12V | Analog measure of system +12 volt main supply | |
| RA3 | A | S4_VREF | Voltage Reference for A/D converter | Unused |
| RA4 | x | | | |
| RA5 | A | S4_VOLTS_N12V | Analog measure of system -12 volt main supply | See S4_VOLTS_P5V |
| RB0 | I/O | S4_PSN_CAN1 | Presence and Serial Number I/O for Canister 1 | These are all lines to one wire serial data EPROMS. See Dallas DS250x data sheet for programming information |
| RB1 | I/O | S4_PSN_CAN2 | Presence and Serial Number I/O for Canister 2 | |
| RB2 | I/O | S4_PSN_CAN3 | Presence and Serial Number I/O for Canister 3 | |
| RB3 | I/O | S4_PSN_CAN4 | Presence and Serial Number I/O for Canister 4 | |
| RB4 | I/O | S4_PSN_CAN5 | Presence and Serial Number I/O for Canister 5 | |
| RB5 | I/O | S4_PSN_CAN6 | Presence and Serial Number I/O for Canister 6 | |
| RB6 | I/O | S4_PSN_CAN7 | Presence and Serial Number I/O for Canister 7 | |
| RB7 | I/O | S4_PSN_CAN8 | Presence and Serial Number I/O for Canister 8 | |
| RC0 | x | | | |
| RC1 | I | S4_ACOK_PS3 | A/C Input OK to Power Supply 3 | Should check only if PSN for power supply indicates presence. |
| RC2 | I | S4_ACOK_PS2 | A/C Input OK to Power Supply 2 | |
| RC3 | I/O | S4_I2C_CLK | Wire Service Bus Data (I2C) | Only used for I2C |
| RC4 | I/O | S4_I2C_CLK | Wire Service Bus Data (I2C) | |
| RC5 | I | S4_ACOK_PS1 | A/C Input OK to Power Supply 1 | See S4_ACOK_PS3 |
| RC6 | O | S4_POWER_ON | Enable main output from power supplies | |
| RC7 | I | S4_POWER_SW | Power On/Off switch (undebounced) | |
| RD0 | I/O | S4_PSN_PS1 | Presence and Serial Number for Power Supply 1 | These are all lines to one wire serial data EPROMS. See Dallas DS250x data sheet |
| RD1 | I/O | S4_PSN_PS2 | Presence and Serial Number for Power Supply 2 | |
| RD2 | I/O | S4_PSN_PS3 | Presence and Serial Number for Power Supply 3 | |
| RD3 | I/O | S4_PSN_BP | Presence and Serial Number for Backplane | Dallas DS250x also |
| RD4 | I/O | S4_PSN_SB | Presence and Serial Number for System Board | Dallas DS250x also |
| RD5 | I | S4_BP_TYPE | Backplane Type ( 0: Small 1: Large ) | |
| RD6 | I/O | S4_TEMP_SCL | Temperature Bus Clock | I2C local bus for temperature probes at different system points |
| RD7 | I/O | S4_TEMP_SDA | Temperature Bus Serial Data | |
| RE0 | I | S4_DCOK_PS3 | D/C Output OK from Power Supply 3 | Should check only if PSN for power supply indicates presence. |
| RE1 | I | S4_DCOK_PS2 | D/C Output OK from Power Supply 2 | |
| RE2 | I | S4_DCOK_PS1 | D/C Output OK from Power Supply 1 | |

Wire Service Canister (System Type ID S5) Processor ID 2x where x is the slot ID

| Pin | Type | Name | Function | Notes |
|---|---|---|---|---|
| RA0 | O | S5_P12V_ENA | Turns on +/- 12 volt to all PCI slots | Used to sequence power to PCI cards. |
| RA1 | O | S5_P5V_ENA4 | Turns on +5 volts to PCI slot 4 | |
| RA2 | O | S5_P5V_ENA3 | Turns on +5 volts to PCI slot 3 | |
| RA3 | O | S5_P5V_ENA2 | Turns on +5 volts to PCI slot 2 | |
| RA4 | O | S5_P5V_ENA1 | Turns on +5 volts to PCI slot 1 | |
| RA5 | ? | | | |
| RB0 | I | S5_CAN_A0 | Canister Address bit 0 | Determine the Wire Service bus address of this canister |
| RB1 | I | S5_CAN_A1 | Canister Address bit 1 | |
| RB2 | I | S5_CAN_A2 | Canister Address bit 2 | |
| RB3 | I | S5_PRSNT_S5 | Special Slot 5 (IOP/PCI jumper) present | Indicates something is in slot 5 |
| RB4 | I/O | S5_PSN_S5 | Present Serial Number for special slot 5 | From DS 250x in slot 5 card (if IOP) |
| RB5 | x | | | |
| RB6 | x | | | |
| RB7 | x | | | |
| RC0 | I | S5_FAN_TP | Tachometer pulse input from selected fan | Generally routed to counter |
| RC1 | O | S5_FAN_SEL0 | Fan Tachometer Multiplexer Select Bit 0 | Select which fan to monitor tach. |
| RC2 | x | | | |
| RC3 | I/O | S5_I2C_CLK | Wire Service Bus Data (I2C) | Only used for I2C |
| RC4 | I/O | S5_I2C_CLK | Wire Service Bus Data (I2C) | |
| RC5 | O | S5_CANFAN_LED | Canister fan fault LED | Assert on any Canister fan failure |
| RC6 | O | S5_CANFLT_LED0 | Canister FRU LED Pin 0 (bicolor LED) | Drive in different combinations for OFF, AMBER, GREEN |
| RC7 | O | S5_CANFLT_LED1 | Canister FRU LED Pin 1 (bicolor LED) | |
| RD0 | I | S5_PRSNT_S1A | PCI card present in Slot 1 (A pin) | PCI slots have 2 presence pins - see PCI spec for usage and meaning. |
| RD1 | I | S5_PRSNT_S1B | PCI card present in Slot 1 (B pin) | |
| RD2 | I | S5_PRSNT_S2A | PCI card present in Slot 2 (A pin) | |
| RD3 | I | S5_PRSNT_S2B | PCI card present in Slot 2 (B pin) | |
| RD4 | I | S5_PRSNT_S3A | PCI card present in Slot 3 (A pin) | |
| RD5 | I | S5_PRSNT_S3B | PCI card present in Slot 3 (B pin) | |
| RD6 | I | S5_PRSNT_S4A | PCI card present in Slot 4 (A pin) | |
| RD7 | I | S5_PRSNT_S4B | PCI card present in Slot 4 (B pin) | |
| RE0 | O | S5_CAN_JTAG | Enable Canister Board JTAG Chain TMS | Required to select the JTAG chain |
| RE1 | O | S5_NMI_S5 | NMI card is special slot 5 (IOP) | Toggle to NMI IOP in slot 5 |
| RE2 | O | S2_FAN_HI | Canister fan speed to high (Assert Hi) | Assert on any Canister fan failure |

Wire Service Remote Interface (System Type ID S6) Processor ID 11

| Pin | Type | Name | Function | Notes |
|---|---|---|---|---|
| RA0 | I/O | S6_PSN_RI | Serial Number information for Remote Interface | |
| RA1 | x | | | |
| RA2 | x | | | |
| RA3 | x | | | |
| RA4 | x | | | |
| RA5 | x | | | |
| RB0 | O | S6_MODEM_DTR | Modem Signal (Data Terminal Ready) | |
| RB1 | I | S6_MODEM_DSR | Modem Signal (Data Set Ready) | |
| RB2 | I | S6_MODEM_CD | Modem Signal (Carrier Detect) | |
| RB3 | I | S6_MODEM_RI | Modem Signal (Ring Indicate) | |
| RB4 | O | S6_MODEM_RTS | Modem Signal (Request To Send) | |
| RB5 | I | S6_MODEM_CTS | Modem Signal (Clear To Send) | |
| RB6 | x | | | |
| RB7 | x | | | |
| RC0 | x | | | |
| RC1 | x | | | |
| RC2 | x | | | |
| RC3 | I/O | S5_I2C_CLK | Wire Service Bus Data (I2C) | Only used for I2C |
| RC4 | I/O | S5_I2C_CLK | Wire Service Bus Data (I2C) | |
| RC5 | x | | | |
| RC6 | O | S6_MODEM_TXD | Modem Signal (Transmit Data ) | Controlled by chip serial interface |
| RC7 | I | S6_MODEM_RXD | Modem Signal (Receive Data ) | |
| RD0 | x | | | |
| RD1 | x | | | |
| RD2 | x | | | |
| RD3 | x | | | |
| RD4 | x | | | |
| RD5 | x | | | |
| RD6 | x | | | |
| RD7 | x | | | |
| RE0 | x | | | |
| RE1 | x | | | |
| RE2 | x | | | |

# Wire Service Network Memory Map

This section defines the Wire Service Network Memory Map for the first Raptor system. Its purpose is to identify all Wire Service addressable entities and describe their function and any special information about them.

This section is incomplete yet and only a small incomplete sample is supplied. (Although some of the more complicated ones are described.)

The address format is "pp:aaaa", where "p" is the processor ID (hexadecimal) of the Wire Service Processor where the data resides and "aaaa" is the hexadecimal address or address range for the data.

| Name | Type | Address | Description | Notes |
|---|---|---|---|---|
| WS_DESC_Pn | STRING | 0n:0000 | Wire Service Processor Type/Description | |
| WS_REV_Pn | STRING | On0001 | Wire Service Software Revision/Date Info | |
| WS_SB_FAN_HI | BIT | 03 | System Board Fans HI | Controls S1_FAN_HI. Set on 0->1 transition of WS_SB_FAN_LED. Cleared by other software |
| WS_SB_FAN_LED | BIT | 03 | System Board Fan Fault LED | Controls S1_SBFAN_LED. It is set whenever any WS_SB_FANFAULTn is set. Log 0->1 transition |
| WS_SB_BUSCORE | BYTE | 03 | System Board BUS/CORE speed ratio to use on reset | Value is asserted on S1_BC_DS[0-3] unless reading DIMM types. Set to 0 on power on. |
| WS_SYS_LCD | STRING | 03 | Value to display on LCD | For a Nx2 display the first N bytes display on top line and the second N bytes display on the bottom line. Manipulates S1_LCD_D[0-7], S1_LCD_RS, S1_LCD_ENA, S1_LCD_RW |
| WS_SB_FAN1 | BYTE | 03 | System Board Fan 1 speed | Approximately every second a fan is selected by S1_FAN_SEL[0-2] and monitored via S1_FAN_TP driving a counter for a known period of time. The counter is then loaded into the appropriate fan speed. If WS_SB_FAN_HI is not set then the speed is compared against WS_SB_FAN_LOLIM. If fan is slow set appropriate WS_SB_FANFAULTn otherwise clear it |
| WS_SB_FAN2 | BYTE | 03 | System Board Fan 2 speed | |
| WS_SB_FAN3 | BYTE | 03 | System Board Fan 3 speed | |
| WS_SB_FAN4 | BYTE | 03 | System Board Fan 4 speed | |
| WS_SB_FANFAULT1 | BIT | 03 | System Board Fan 1 Faulted | |
| WS_SB_FANFAULT2 | BIT | 03 | System Board Fan 2 Faulted | |

| | | | | |
|---|---|---|---|---|
| WS_SB_FANFAULT3 | BIT | 03 | System Board Fan 3 Faulted | |
| WS_SB_FANFAULT4 | BIT | 03 | System Board Fan 4 Faulted | |
| WS_SB_FAN_LOLIM | BYTE | 03 | Fan speed low speed fault limit | Set to ??? on power on |
| WS_SB_DIMM_SEL | BYTE | 03 | The DIMM select bits to use when reading DIMM_TYPE | The low order 4 bits are the select bits to use when WS_SB_DIMM_TYPE is read. |
| WS_SB_DIMM_TYPE | BYTE | 03 | The type of DIMM in the DIMM_SEL position | When read asserts value of WS_SB_DIMM_SEL on S1_BC_DS[0-3] and then returns value of S1_DIMM_D[0-7]. |
| WS_SB_FLASH_ENA | BIT | 04 | Indicates FLASH ROW write enabled | Set/Cleared by debounced 0->1 transition of S2FLASH_SW.<br>Controls state of S2_FLASH_WE and S2_FLASH_LED. |
| WS_SB_FRU_FAULT | BIT | 04 | Indicates the FRU status | At power on starts at 1. Controls S2_SBFLT_LED[0-1] for bicolor LED colors 0=Green 1=Amber, Cleared by other software |
| WS_SYS_OVERTEMP | BIT | 04 | Indicates Overtemp fault | At power on is set. Controls S2_OVRTMP_LED. Controlled by wire service backplane processor. |
| WS_SB_JTAG | BIT | 04 | Enables JTAG chain on system board | Clear at power on. Controls S2_SB_JTAG |
| WS_SB_CPU_PRES | BYTE | 04 | CPU Presence bits (LSB = CPU1) | Assemble from S2_PRES_CPU[1-4] |
| WS_SB_CPU_ERR | BYTE | 04 | CPU Error bits (LSB = CPU1) | Assemble from S2_ERROR_CPU[1-4] |
| WS_SB_CPU_TEMP | BYTE | 04 | CPU Thermal fault bits (LSB = CPU1) | Assemble from S2_TEMP_CPU[1-4} |
| WS_SB_CPU_POK | BYTE | 04 | CPU Power OK (LSB = CPU1) | Assemble from S2_POK_CPU[1-4] |
| WS_NMI_MASK | BYTE | 04 | CPU NMI processor mask (LSB=CPU1) | Defaults to all ones on power up |
| WS_NMI_REQ | BIT | 04 | NMI Request bit | When set pulse S2_NMI_CPUn corresponding to each bit set in WS_NMI_MASK. Then clear request bit. Log Action |
| WS_SYSFAULT | BIT | 04 | System Fault Summary | This bit is set if any faults detected in the system. Controls S2_SYSFLT_LED<br>Bits scanned WS_SP_CPU_FAULT, WS_SB_FRU_FAULT ( other faults?) |
| WS_SB_CPU_FAULT | BIT | 04 | CPU Fault Summary | This bit is set if (( WS_SB_CPU_ERR I WS_SB_CPU_TEMP I -WS_SB_CPU_POK ) & -WS_SB_CPU_PRES) != 0. Log 0->1 transition with CPU bytes. |

| | | | | |
|---|---|---|---|---|
| WS_BP_P5V | BYTE | 02 | Analog Measure of +5 volt main supply | read from S4_VOLTS_P5v |
| WS_BP_P3V | BYTE | 02 | Analog Measure of +3.3 volt main supply | read from S4_VOLTS_P3v |
| WS_BP_P12V | BYTE | 02 | Analog Measure of +12 volt main supply | read from S4_VOLTS_P12v |
| WS_BP_P5V | BYTE | 02 | Analog Measure of -12 volt main supply | read from S4_VOLTS_N12V |
| WS_SYS_CAN_PRES | BYTE | 02 | Presence bits for canisters (LSB=1, MSB=8) | controlled by S4_PSN_CAN[1-8]. A previous value byte needs to be maintained so canister transitions can be recognized. Previous value initialized to zero. Periodic monitor scans for new canisters. When new canister is recognized read full serial data and store in WS_SYS_CAN_SERIALn then log and send event |
| WS_SYS_PS_PRES | BYTE | 02 | Presence bits for power supplies (LSB=1, MSB=3) | controlled by S4_PSN_PS[1-3]. A previous value byte needs to be maintained so power supply transitions can be recognized. Previous value initialized to zero. Periodic monitor scans for new power supplies. When new power supply is recognized read full serial data and store in WS_SYS_PS_SERIALn then log and send event |
| WS_SYS_PS_ACOK | BYTE | 02 | Power supply ACOK status (LSB=1, MSB=3) | controlled by S4_ACOK_PS[1-3]. A previous value byte needs to be maintained so power supply transitions can be recognized. Previous value initialized to zero. Periodic monitor scans for changes is ACOK and sends events |
| WS_SYS_PS_DCOK | BYTE | 02 | Power supply DCOK status (LSB=1, MSB=3) | controlled by S4_DCOK_PS[1-3]. A previous value byte needs to be maintained so power supply transitions can be recognized. Previous value initialized to zero. Periodic monitor scans for changes is ACOK and sends events |
| WS_SYS_BP_TYPE | BYTE | 02 | Type of system backplane currently only two types Type 0 = 4 canister (small) and Type 1= 8 canister (large) | controlled by S4_BP_TYPE |
| WS_SYS_TEMP_SB1 | BYTE | 02 | Temperature of system board position 1 | controlled by reading Dallas temperature transducers connected to serial bus on S4_TEMP_SDA and S4_TEMP_SCL |
| WS_SYS_TEMP_SB2 | BYTE | 02 | Temperature of system board position 2 | |
| WS_SYS_TEMP_BP1 | BYTE | 02 | Temperature of backplane position 1 | |
| WS_SYS_TEMP_BP2 | BYTE | 02 | Temperature of backplane position 2 | |

| | | | | |
|---|---|---|---|---|
| WS_SYS_TEMP_WARN | BYTE | 02 | Warning temperature. Initialized to ??? | If any WS_SYS_TEMP_xxn exceeds this value, log, send event, set WS_SYS_OVERTEMP |
| WS_SYS_TEMP_SHUT | BYTE | 02 | Shutdown temperature. Initialized to ??? | If any WS_SYS_TEMP_xxn exceeds this value, log and clear WS_SYS_POWER |
| WS_SYS_REQ_POWER | BIT | 02 | Set to request main power on | |
| WS_SYS_POWER | BIT | 02 | Controls system master power S4_POWER_ON | When this bit is set 0->1 set S4_POWER_ON, WS_SYS_RUN = 0, WS_SYS_RSTIMER = 5 and log. When this bit is cleared clear S4_POWER_ON and log. |
| WS_SYS_RSTIMER | BYTE | 02 | Used to delay reset/run until power stabilized | Counts down to 0 at 10 counts per second. When 1->0 transition sets WS_SYS_RUN. |
| WS_SYS_RUN | BIT | 02 | Controls the system halt/run line S1_OK_TO_RUN. | If this bit is cleared, clear S1_OK_TO_RUN and log.<br>If this bit is set, set S1_OK_TO_RUN and log. |
| WS_CAN_POWER | BIT | 2x | Controls canister PCI slot power | When set then set S5_P5V_ENA[1..4],S5_P12V_ENA in that order with small (about 1 ms) delay between each, then log.<br>When cleared then clear S5_P12V_ENA, S5_P5V_ENA[1..4] then log |
| WS_CAN_PCI_PRESENT | BYTE | 2x | Reflects PCI card slot[1..4] presence indicator pins ( MSB to LSB) 4B,4A,3B,3A,2B,2A,1B, 1A | Reflects data from S5_PRSNT_S[1..4][A/B) |
| WS_CAN_S5_PRESENT | BIT | 2x | Indicates the presence of something in slot 5 | Reflects S5_PRSNT_S5 |
| WS_CAN_S5_SMART | BIT | 2x | Indicates something other than a passive board in slot 5 | On power up attempt to read Dallas serial number chip using S5_PSN_S5. If present set this bit and read full serial data and store in WS_SYS_CAN_IOP_SERIALn |
| WS_CAN_FAN_HI | BIT | 2x | Canister Fans HI | Controls S1_FAN_HI. Set on 0->1 transition of WS_SB_FAN_LED. Cleared by other software |
| WS_CAN_FAN_LED | BIT | 2x | Canister Fan Fault LED | Controls S5_CANFAN_LED. It is set whenever any WS_CAN_FANFAULTn is set. Log 0->1 transition |
| WS_CAN_FANFAULT1 | BIT | 03 | Canister Fan 1 Faulted | |
| WS_CAN_FANFAULT2 | BIT | 03 | Canister Fan 2 Faulted | |

| | | | | |
|---|---|---|---|---|
| WS_CAN_FAN1 | BYTE | 2x | Canister Fan 1 speed | Approximately every second a fan is selected by S5_FAN_SEL0 and monitored via S5_FAN_TP driving a counter for a known period of time. The counter is then loaded into the appropriate fan speed. If WS_CAN_FAN_HI is not set then the speed is compared against WS_CAN_FAN_LOLIM. If fan is slow set appropriate WS_ÇAN_FANFAULTn otherwise clear it |
| WS_CAN_FAN2 | BYTE | 2x | Canister Fan 2 speed | |
| WS_CAN_FAN_LOLI M | BYTE | 2x | Fan low speed fault limit | Set to equivalent of xxx RPM on power on |
| WS_CAN_JTAG_ENA | BIT | 2x | Enable JTAG TMS chain for canister | Copy set value to S5_CAN_JTAG |
| WS_CAN_NMI_S5 | BIT | 2x | NMI card in slot 5 | when set, pulse S2_NMI_S5 |
| WS_RI_CD | BIT | 11 | Status of Remote Port Modem CD | Follows S6_MODEM_CD |
| WS_RI_DTR | BIT | 11 | State of Remote Port Modem DTR | Controls S6_MODEM_DTR |
| WS_RI_DSR | BIT | 11 | Status of Remote Port Modem DSR | Follows S6_MODEM_DSR |
| WS_RI_RTS | BIT | 11 | Status of Remote Port Modem RTS | Controls S6_MODEM_RTS |
| WS_RI_CTS | BIT | 11 | Status of Remote Port Modem CTS | Follows S6_MODEM_CTS |
| WS_RI_CALLOUT | BYTE | 11 | Controls Call out Script activation | If written to it initiates Call out sequence programmed in WS_SYS_CALL_SCRIPT passing value as argument to script. Log it ( Format of Script Programs TBD ) |
| WS_RI_EVENTS | EVENT | 11 | Remote Interface Event Queue | See Event Data type description in prior section. |
| WS_SI_EVENTS | EVENT | 10 | System Interface Event Queue | See Event Data type description in prior section. |
| WS_SYS_LOG | LOG | 01 | System Log | The system log kept in NVRAM ( See LOG data type in previous section ) |
| WS_SYS_SCREEN | SCREE N | 01 | System Screen | A copy of the most recent character mode screen from the system video display ( See SCREEN data type in previous section ) |
| WS_SYS_SB_SERIAL | STRING | 01 | Last known System Board serial data | |
| WS_SYS_BP_SERIAL | STRING | 01 | Last known Back Plane serial data | |
| WS_SYS_RI_SERIAL | STRING | 01 | Last known Remote Interface serial data | |

| | | | | |
|---|---|---|---|---|
| WS_SYS_CAN_SERIAL[1-8] | STRING | 01 | Last known Canister [1-8] Serial data | May be zero length if no canister ever seen |
| WS_SYS_IOP_SERIAL[1-8] | STRING | 01 | Last known IOP in Canister [1-8] Serial data | May be zero length if no canister ever seen or current canister has no IOP |
| WS_SI_QUEUE | QUEUE | 01 | Queue of data going to System Interface | See Queue data type in previous section |
| WS_RI_QUEUE | QUEUE | 01 | Queue of data going to Remote Interface | See Queue data type in previous section |
| WS_SYS_XDATA | BYTE ARRAY | 01 | Byte Array for storage of arbitrary external data in NVRAM | Wire Service just maintains this data area and is unaware of the meaning of any data stored in it. |
| WS_SYS_EXT_KB | BYTE | 01 | Size of the WS_SYS_XDATA in kilobytes | Necessary for memory management of the data area |

# Wire Services Processor Functions

The previous section either states or directly implies many monitoring, control and feedback operations carried out by the different wire service processors. In addition the following functions or actions must be implemented:

(Still rough)

1) Monitor S1_RESET_SW and debounce and keep state

2) Startup process

3) reset process

4) Monitor S2_FLASH_SW

5) Monitor temperatures (backplane processor) control WS_SYS_OVERTEMP

6) Monitor S2_NMI_SW and debounce - set WS_NMI_REQ on 0->1 transition

7) Monitor fault conditions for system fault and control WS_SYSFAULT_LED

8) Monitor S4_PSN_CANn and S4_PSN_PSn and log any transitions and signal events.

9) On backplane processor power on read system board serial information on S4_PSN_SB and store it in WS_SYS_SB_SERIAL and backplane board serial on S4_PSN_BP and store it in WS_SYS_BP_SERIAL.

10) Note that canister Wire Service processor address is determined by taking the canister addresss S5_CAN_A[0-2] and adding 0x20 to create the address.

11) Remote on power up and occasionally after reads serial data from S6_PSN_RI and stores it in WS_SYS_RI_SERIAL.

12) Events are always sent to both WS_SI_EVENTS and WS_RI_EVENTS with no retry if no processor response from target.

## Wire Service and Raptor BIOS Interactions

The following are items which it is known that the BIOS must implement.

- BIOS must determine current DIMM configuration by setting WS_SB_DIMM_SEL appropriately for each DIMM slot and reading WS_SB_DIMM_TYPE to determine DIMM type and then validating that this is an OK configuration. Also, memory can be initially sized this way.
- BIOS can read WS_SB_CPU_PRES To determine which processors are present and then go out on the bus and somehow get the processor type and speed information to determine the correct BUS/CORE speed ratio. Once that is determined. The BIOS reads WS_SB_BUSCORE to determine the actual BUS/CORE speed ratio and if it is incorrect, write the correct one to WS_SB_BUSCORE, set WS_SYS_RSTIMER to 5 ( 5/10 second ), and clear WS_SYS_RUN. This will reset the system using the new BUS/CORE ratio. ( Note: If WS_SYS_RSTIMER is not set, wire service will not set WS_SYS_RUN after the BIOS clears it and the system will remain halted.
- The BIOS must read WS_SYS_CAN_PRES to determine which canisters are present and then set WS_CAN_POWER on each canister to enable power to the PCI cards before configuring the PCI busses.
- All WS_xx_FRU_LEDs are amber at power up. The BIOS sets WS_xx_FRU_LED to green when satisfied that the system board or canister is ok. Diagnostics may turn WS_xx_FRU_LEDs to amber on diagnostic failure fault.

## Wire Service Issues Needing Resolution

1) Does Fan Fault roll up into system fault LED?

2) Once a fan faults at low speed (i.e. drops below low speed limit), the fan fault LED is set and fans go to high speed. How is faulting fan identified? Look at system NVRAM log?) How is fault reset on fan swap? Can fans fault at high speed? ( hi speed is a result of a fan fault)

3) How should power on work when A/C power returns after being gone?

4) If there is a CPU thermal fault should the over temp LED turn on or system board fault LED?

5) How much monitoring/logging should be done for CPU fault signals?

6) What are all the conditions that turn on system fault summary LED?

7) Are there any times that the system fault summary must be valid? May not be valid? Response times?

8) OverTemp lamp goes on at over temp warning limit. Power is turned off if shutdown limit is reached. If we shutdown the box for overtemp reason, how is overtemp cleared?

## Wire Service Remote Interface Serial Protocol

To be Specified

## Wire Service Call out Script Syntax Definition

To be specified